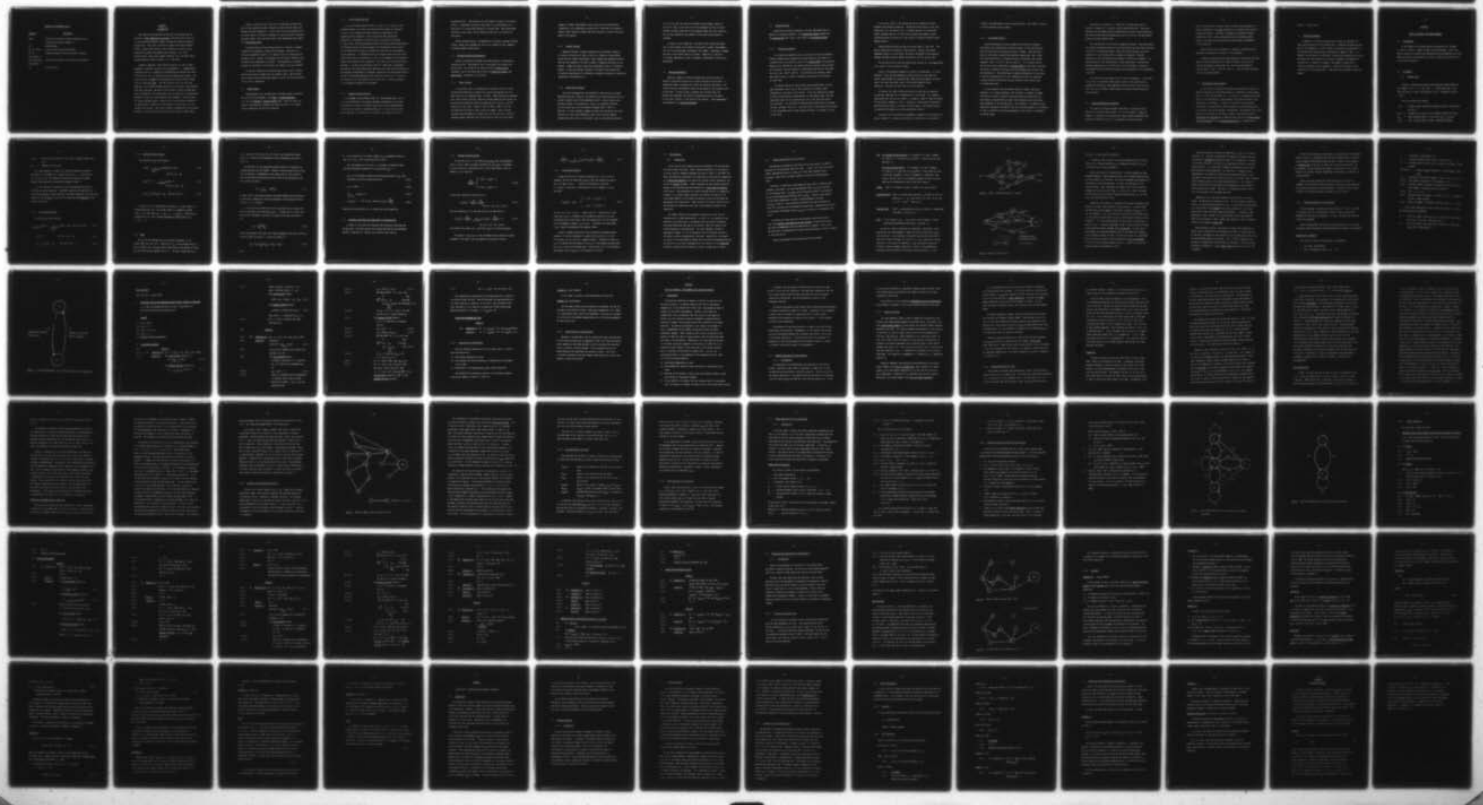
AD-A068 138 MASSACHUSETTS INST OF TECH CAMBRIDGE LAB FOR INFORMA--ETC F/G 17/2
FAILSAFE DISTRIBUTED OPTIMAL ROUTING IN DATA-COMMUNICATION NETW--ETC(U)
MAR 79 M SIDI, A SEGALL N00014-75-C-1183
UNCLASSIFIED LIDS-R-890 NL

1 OF 2
AD A068138



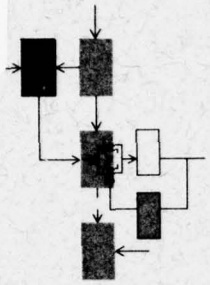
LEVEL II

March, 1979

LIDS-R-890

Research Supported By:

ARPA Contract N00014-75-C-1183
ONR Contract ONR/N00014-77-C-0532



FAILSAFE DISTRIBUTED OPTIMAL ROUTING IN DATA-COMMUNICATION NETWORKS

M. Sidi
A. Segall

Laboratory for Information and Decision Systems
Formerly
Electronic Systems Laboratory
MASSACHUSETTS INSTITUTE OF TECHNOLOGY, CAMBRIDGE, MASSACHUSETTS 02139

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)

FAILSAFE DISTRIBUTED OPTIMAL ROUTING IN DATA-COMMUNICATION NETWORKS,

5. TYPE OF REPORT & PERIOD COVERED

6. PERFORMING ORG. REPORT NUMBER

LIDS-R-890

7. AUTHOR(s)

M. Sidi
A. Segall

8. CONTRACT OR GRANT NUMBER(s)

Contract ONR/N00014-75-C-1183

9. PERFORMING ORGANIZATION NAME AND ADDRESS

M.I.T.
Laboratory for Information and Decision Systems
Cambridge, MA 02139

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

Program Code No. 5T10
ONR Identifying No. 049-383

11. CONTROLLING OFFICE NAME AND ADDRESS

Defense Advanced Research Projects Agency
1400 Wilson Boulevard
Arlington, Virginia 22209

12. REPORT DATE

March 1979

13. NUMBER OF PAGES

152

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)

Office of Naval Research
Information Systems Program
Code 437
Arlington, Virginia 22217

15. SECURITY CLASS. (of this report)

UNCLASSIFIED

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

15 N00014-75-C-1183
N00014-77-C-0532

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Iterative protocols for adaptive routing in line and message switched data communication networks are presented in this thesis. The protocols have the following features:

1. Distributed computation is used in the sense that each node in the network bases all its decisions on control messages received only from its neighbors. Thus, each node in the network determines individually onto which of its outgoing links to send the flow, addressed to a specific destination. The control messages exchanged between neighbors contain information about network

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

connectivity, network congestion and link failures.

2. Loop-free routing for each destination is maintained in the network at all times Generally, prevention of loops results in saving resources and reduction in delay. In addition, loop-free routing establishes a partial ordering on the set of nodes of the network. The latter property is extensively utilized throughout this work.

3. Failsafe and deadlock-free operation of the protocols is guaranteed, meaning that after arbitrary failures and additions of links and nodes, the network recovers in finite time. Recovery means that routing paths are provided between all connected nodes.

4. For stationary input traffic statistics and fixed topology the protocols are optimal. They reduce network delay at each iteration and minimum average delay over all routing assignments is obtained in steady-state.

Proofs of all features are provided.

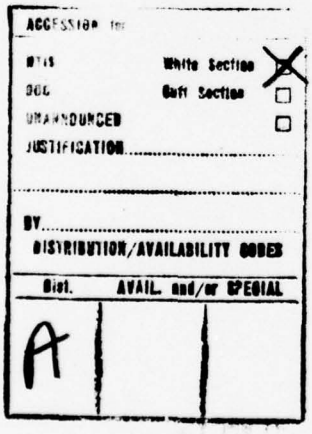
The protocols are intended for quasi-static applications where the input requirements are slowly changing with time and where occasionally links or nodes fail or are added to the network.

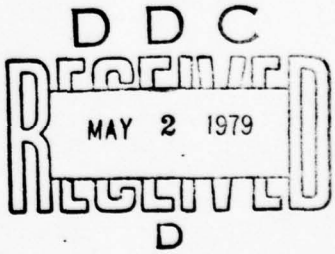
March 1979 LIDS-R-890

FAILSAFE DISTRIBUTED OPTIMAL ROUTING IN DATA-COMMUNICATION NETWORKS

by

M. Sidi*
A. Segall**





* Technion Institute of Technology, Haifa, Israel

** Technion Institute of Technology, Haifa, Israel
and
Consultant, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, Mass. 02139

The work of A. Segall was supported in part by the Advanced Research Project Agency of the US Department of Defense (monitored by ONR) under Contract No. N00014-75-C-1183, and in part by the Office of Naval Research, Contract No. ONR/N00014-77-C-0532.

CONTENTS

CONTENTS (Cont.)

CONTENTS (Cont.)

ABSTRACT

Iterative protocols for adaptive routing in line and message switched data communication networks are presented in this thesis. The protocols have the following features:

1. Distributed computation is used in the sense that each node in the network bases all its decisions on control messages received only from its neighbors. Thus, each node in the network determines individually onto which of its outgoing links to send the flow, addressed to a specific destination. The control messages exchanged between neighbors contain information about network connectivity, network congestion and link failures.

2. Loop-free routing for each destination is maintained in the network at all times. Generally, prevention of loops results in saving resources and reduction in delay. In addition, loop-free routing establishes a partial ordering on the set of nodes of the network. The latter property is extensively utilized throughout this work.

3. Failsafe and deadlock-free operation of the protocols is guaranteed, meaning that after arbitrary failures and additions of links and nodes, the network recovers in finite time. Recovery means that routing paths are provided between all connected nodes.

4. For stationary input traffic statistics and fixed topology the protocols are optimal. They reduce network delay at each iteration and minimum average delay over all routing assignments is obtained in steady-state.

Proofs of all features are provided.

The protocols are intended for quasi-static applications where the input requirements are slowly changing with time and where occasionally links or nodes fail or are added to the network.

Proofs of all features are provided.

The protocols are intended for quasi-static applications where the input requirements are slowly changing with time and where occasionally links or nodes fail or are added to the network.

GLOSSARY OF NOTATIONS

NOTATION	DEFINITION
N	- number of nodes in a network.
L	- set of links in a network.
(i,k)	- directed link from node i to node k.
$r_i(j)$	- average traffic entering the network at node i and destined for node j.
$f_{ik}(j)$	- average flow in link (i,k) of traffic destined for node j.
f_{ik}	- total average traffic in link (i,k).
f	- the set of link flows.
$t_i(j)$	- total average traffic at node i destined for node j.
$\phi_{ik}(j)$	- fraction of the node flow $t_i(j)$ that is routed through link (i,k).
C_{ik}	- capacity of link (i,k).
D_{ik}	- average delay per unit time of all traffic sent over link (i,k).
D_T	- total delay in the network per time unit.
$1/\mu$	- average unit of traffic length.
P_{ik}	- propagation delay in link (i,k).
K_{ik}	- nodal processing time in node k.
$d_i(j)$	- estimated marginal delay of node i from destination j.
SINK	- destination node.
d_i	- estimated marginal delay of node i from SINK.
b_i	- blocking status of node i.
n_i	- current counter number of node i.

GLOSSARY OF NOTATIONS (Cont.)

NOTATION	DEFINITION
MSG(m,d,b,ℓ)	- updating message sent by node ℓ.
FAIL(ℓ)	- failure detected on link (i,ℓ).
WAKE(ℓ)	- link (i,ℓ) becomes operational.
$F_i(\ell)$	- status of link (i,ℓ) as seen from node i.
$N_i(\ell)$	- the number m received from neighbor ℓ during the current iteration.
$D'_{i\ell}$	- estimated (or calculated) marginal delay on link (i,ℓ).
$D_i(\ell)$	- sum of $D'_{i\ell}$ and last number d received at i from neighbor ℓ.
$B_i(\ell)$	- blocking status of neighbor ℓ as known at i.
$Z_i(\ell)$	- a synchronization number indicating the iteration upon which the link (i,ℓ) can be brought up.
$R_i(\ell)$	- status of neighbor ℓ (being a son).
mx_i	- the largest number m received by i up to the current time from all neighbors.
p_i	- preferred son of node i.
CT	- a flag indicating the number of transitions the Finite-State-Machine has already performed triggered by the current message.
REQ(m)	- request message destined for SINK to start an iteration with counter number (m+1).
C_i, A_i	- sets of neighbors of node i.
SON_i	- set of all sons of node i.
η	- a parameter.

GLOSSARY OF NOTATIONS (Cont.)

NOTATION	DEFINITION
PC(m)	- instant of occurence of proper completion of an iteration with counter number m.
RG	- routing graph.
S1, S2, $\tilde{S2}$, S3	- states of the Finite-State-Machine.
C1, $\tilde{C2}$	- changes performed in the Finite-State- Machine.
T12, T13, T21, T22, T23, $\tilde{T22}$, T32, $\tilde{T22}$, $\tilde{T23}$	- transitions performed in the Finite-State-Machine.
s_i	- state of node i.

CHAPTER 1

INTRODUCTION

Many efforts have been and are devoted to the design and the analysis of data communication networks, which provide the facility of interconnection between a number of users for sharing resources between them. This kind of networks includes time shared computer systems, medical data networks, bank transaction systems, airline reservation systems, multipurpose data networks, (e.g. AT&T, Western Union), large scale computer networks (e.g. the ARPA network - Advanced Research Projects Agency), etc. [SCHW 72a].

Generally speaking, a data network consists of a set of users (computers, terminals, displays, etc.)connected by a communication subnetwork that is in charge with transferring data between the users. In this work we will be concerned with the communication subnet. The latter consists of nodes which exchange data with each other through a set of connecting links. The nodes (IMP-in ARPA) are real-time computers, with limited storage and processing resources, which perform some basic functions, the main of which being to direct the data that passes through them. The connecting links are some type of communication channels of relatively high bandwidth and reasonably low error rate. The subnet topology design is usually one of the difficult problems in the design phase. However, for the purposes of this work, we do not consider this problem , and assume a general, geographically distributed topology, in which each node can have multiple paths to other nodes.

Clearly, there must exist some set of disciplines governing the flow of data between the users, between the users and the nodes, and between the nodes themselves. In this work we are only concerned with the rules used by the nodes to determine in which directions to deliver the data traffic, from the source node to the destination node, namely with the routing policy.

The complication of the routing problem in a network is commonly a question of assumptions, formulations and goals, involved in it. The more assumptions we make, it is expected the less complicated the problem will be. However, the designer and the analyst certainly wish to make as few assumptions as possible. The formulation is probably a matter of convenience, and the goals can differ in various problems.

In the next subsections we describe some network types, routing policies and control schemes that are commonly used in data networks. Also an outline of the following chapters is given and the contribution of this work is emphasized.

1.1: Network Types

Corresponding to any routing policy, two basic types of networks are in use or in development - the line or circuit switching type, and the message or packet switched type. These two types are distinct techniques for communication among the nodes of the subnet, and any combination of the two is possible.

1.1.1: Line Switched Networks

In a line switched network [TYM 71], which is very similar to the telephone network, the source and the destination nodes are connected by one or more communication paths that are established at the beginning of the connection, and are cancelled when the desired connection is terminated or when the path is disrupted by failures. In other words, the connecting paths between the source and the destination are dedicated before any data messages are transmitted from the source to the destination through the selected path and exist for the duration of the connection. In different routing strategies these paths may either remain fixed until the connection is over, or be changed, but not cancelled, during the existence of the connection. One version of the line-switching strategy is "virtual line-switching" [TYM 71], where data is forwarded according to the established paths connecting the source and the destination, but messages corresponding to different connections are multiplexed together on each link and demultiplexed on the other end of the link. In this way, the portion of the link capacity used by each call is varying according to its momentary transmission requirement.

1.1.2: Message Switched Networks

In a message switched network [MCQ 77], each message makes its own way to the destination, and usually messages corresponding to the same destination will travel on different paths which are not predetermined. In this type of network, a message entering it, is first stored in the source node until its time comes to be sent on an outgoing link to a

neighboring node. (The selection of the neighbor is exactly the routing policy). Having been received by that node, it is stored again in a queue until it is being sent forward to the next node. Thus the message continues to pass links, and be queued at nodes until it reaches its destination.

Packet switched network is fundamentally the same as message switched network, except that messages are split into a number of small segments of maximum length called packets.

1.2: Routing Policies Classification

Several classification schemes have been proposed to characterize routing policies. The scheme we use is according to how dynamic the policies are. On one end of the scale we have the purely static strategies, and on the other end we have the completely dynamic ones. Quasi-static strategies lie in between.

1.2.1: Static Routing

In the purely static or deterministic situation, the set of rules dictating the fractions of traffic with a given destination sent by a node to each of its outgoing links, is fixed. These fractions are decided upon, under several criteria, before the establishment of the network, by making various assumptions about the node and link locations, and the capacities of the links. [FRAN 71, CHO 72, FRA 73, GER 73, CAN 74]. The decisions are fixed in time, and do not change. Static routing strategies are non-adaptive in nature and lack the ability to cope with changing network conditions such as failures of nodes and links and/or

changes in traffic requirements, and as such are too unreliable and inefficient to be considered in practice for nontrivial size networks. However, their simplicity makes them very attractive to use at the design phase of the network.

1.2.2: Dynamic Routing

Completely dynamic routing strategies allow continuous changing of routes as a function of time, as well as a function of the network states such as traffic requirements, queue lengths and component failures. They are thus supposed to be able to adapt to changing conditions in the network. Dynamic routing is much more advantageous since it is adaptive. However, it has some inherent drawbacks, the main of which being that it requires large amounts of overhead per message for purposes of adressing, reordering at destinations etc.

1.2.3: Quasi-Static Routing

Given the advantages and the drawbacks of each of the two already described policies, naturally, one should try to devise policies that can possibly acquire some of the advantages of both. Using a quasi-static routing strategy is one possiblity, since it is adaptive in nature, but the routes can not be continuously changed [GALL 77,SEG 77a, SEG 77b]. In this strategy, changes of routes are allowed only at given intervals of time, and/or whenever a need to do so arises because extreme situations occur in the network, such as link and node failures

or recoveries. The time intervals between routing changes should be relatively long, so that most of the time messages are sent in order, causing a serious reduction in the overhead needed, but they should not be too long, otherwise, the inferior of the fixed routing would be revealed.

In order to allow adaptivity, the quasi-static routing procedure has to sense changes in the network status and in traffic requirements, and then to route messages accordingly, for example, congested or damaged portions of the network should be avoided. Adaptivity to failures is of great importance in order to maintain a good grade of service for the network.

1.3: Routing Information

Generally, adaptive routing strategies base their decisions on measured values which describe the salient features of the network. In completely dynamic strategies the values are measured continuously, and actually are the instantaneous states of the queues at the outgoing links of the nodes. In quasi-static strategies the varying values are periodically measured, and consist of quantities such as the queues at the links, traffic, or the status of the network. These measurements are reffered to as routing information.

1.4: Networks Control

Gathering the routing information, two main approaches exist to conduct the routing procedures - the centralized control scheme and the decentralized one, which is also known as distributed control.

1.4.1.: Centralized Control

In a centralized adaptive policy, the nodes collect the necessary routing information for making the routing decisions, and send it to a special node in the network, which is the central node or the governor. Receiving the information, the central node has a global status picture of the network, and can dictate its routing decisions back to the nodes for actual use [TYM 71, BRO 75]. The decisions are naturally based upon some criteria, in order to optimize the routing in the network in some sense.

The centralized policy seems simple and straightforward, and has some advantages, mainly due to the availability of global status information at one place in the network. Since the computations needed to make the decisions are conducted only by the central node, the used algorithms might be very sophisticated, and at the same time simple to understand. It is possible to achieve several goals such as "optimal" routing, avoiding "loops" etc. Also the nodes in the network are relieved of the troublesome task to make routing decisions, so overhead is saved at each node.

In practice, however, the centralized control scheme has several drawbacks and inherent weaknesses. Should the central node, or the links connecting it to the network, fail, or should some part of the network become isolated, then all or some of the network nodes remain without routing decisions for actual use, and a part or the entire network cannot operate anymore.

Another possible difficulty may arise when nodes or links fail. The central node must be notified of such failures. However, the failed components might lie on the paths, previously determined by the governor, between the nodes trying to report the failure and the central node.

We also notice that since the central node conducts all the computations, it is likely to be very heavily loaded.

Finally, the unbalanced demands on network link bandwidth, is a clear drawback. Since routing information and decisions go to and from the central node through its outgoing links, these links are heavily utilized, when at the same time other links in the network might be bored. Apparently, this may also limit the size of the network.

Of course, the simple centralized control scheme might be improved in such ways that some of its weaknesses will be overcome. For instance, the governor may have back-up centers, on stand-by, ready to take the control of the network, whenever it fails. Eventually, there arises the problem of identifying which node is in control of which nodes. Such identification is essential for proper work of any centralized control.

A natural way to overcome the fundamental weaknesses of the centralized control scheme is to wonder why shouldn't all the nodes in the network be

"centers" and participate in the routing decisions. This leads us directly to the distributed control scheme.

1.4.2: Distributed Control

Distributed adaptive control schemes have neither the inherent inefficiency and unreliability of fixed routing, nor the unreliability and size limitations of centralized control schemes. Here each node needs to individually perform the necessary computations, and to make the routing decisions in collaboration with its adjacent nodes called neighbors. [STE 77, GALL 77, SEG 77a, MCQ 77]. It is usually done by storing the routing information in routing tables at each node, and using the tables to identify the output link each message has to select, for each destination. The tables might be updated periodically or only when it matters (asynchronously) or any combination of both, by using the routing information each node collects internally and receives from its neighboring nodes.

In most commonly used distributed adaptive schemes, each node estimates, by a certain procedure, the "distance" it expects a message would have to travel in order to reach each possible destination, if the message is transmitted over each of the outgoing links, and stores these estimates in the routing table. The "distance" is a measure, which numerically expresses the quantity that the routing procedure is to minimize in order to achieve the desired performance of the network, as defined at the design stage.

Each node in a network of N nodes has a routing table which is typically composed of N-1 entries, one for each destination. Each entry indicates the estimated minimal distance from this node to each destination and also the next node the message must pass on its way to the destination, along the minimal distance path.

The routing table of each node is updated as follows. Each node selects the minimal estimated distance for each destination and sends these estimates to each of its neighboring nodes. Receiving these estimates, each node constructs its own routing table by adding its neighbors' received estimates, to its own estimates of distance to each of its neighbors. For each destination, the routing table is then constructed to indicate the selected outgoing link, for which the sum of the estimated distance to the neighbor and the estimated distance from the neighbor to the destination, is minimal.

Distributed routing schemes are not lacked of weaknesses. Since there is no place in the network where global status of the network and its topology are available, then temporary "loops" may exist within the network, and also it becomes twofold harder to maintain failsafe operation of the network.

1.5: Routing Performance Evaluation

Any specific routing assignment algorithm is to achieve certain goals, and to fulfill some criteria. It is to be simple, to adapt to changes, to converge to an accurate and stable routing assignment under stationary conditions, and it is to optimize some cost functions.

The cost function most commonly used to evaluate the performance of a routing algorithm is the delay experienced by messages when traveling through the network. The delay is composed of propagation delays, transmission delays, nodal processing delays and queueing delays. Clearly, the delay must be minimized for good grade of performance of the algorithm.

Other criteria may be reliability, throughput, which are to be maximized, and network cost which is to be minimized.

A number of algorithms have been proposed to achieve some of these goals for static routing [FRA 73, GER 73, CAN 74], as well as for centralized adaptive routing [BRO 75], and distributed adaptive routing [STE 77, GALL 77, SEG 77a, NAYL 77]. Some of these algorithms will be discussed presently.

1.6: Contribution of this Thesis

In this thesis we develop distributed routing protocols which are natural extensions of three known protocols introduced in recent papers [GALL 77], [SEG 77a] and [SEG 77c]. In [SEG 77c] a failsafe distributed protocol which maintains a single optimal route from each node to the destination is developed. In [GALL 77] and [SEG 77a] quasi-static distributed routing protocols which minimize the total expected delay in a network with fixed topology, are proposed. The features of the above protocols are unified in our thesis, namely our protocols are both distributed and failsafe and in addition they indicate the exact amounts of flow splitting, so that minimum average delay is obtained in the

network in steady-state.

1.7: Outline of Chapters

In Chapter 2 we present the protocols of [GALL 77] and [SEG 77a] that provide the basis of the present work. A different presentation from that in the references is used in order to facilitate the explanation of our failsafe distributed protocols described in Chapters 3 and 4. In Chapter 3 our protocols are described in detail, their properties are stated and proofs for these properties are given in Appendix A and Appendix B. The protocol is completed in Chapter 4 and Chapter 5 deals with a simulation program, given in Appendix C, which was developed to check the protocol of individual nodes. Conclusions are discussed in the final Chapter.

CHAPTER 2

ROUTING PROTOCOLS FOR FIXED TOPOLOGY

2.1: Introduction

In this chapter the routing protocols proposed by R.G. Gallager in [GALL 77] and by A. Segall in [SEG 77a] are described. First, two network models are presented and some definitions and equations are stated. Then the protocols are described by using a different presentation than in the above references.

2.2: The Models

2.2.1: General Model

Consider a communication network consisting of N nodes denoted by the integers $\{1,2,3,\ldots,N\}$, and a set L of directed links. Let a link from node i to node k be denoted by (i,k). An example is given in Fig. 1.

Let us now define some symbols:

$r_i(j)$ = average traffic entering the network at node i and destined for node j.

$f_{ik}(j)$ = average flow in link (i,k) of traffic destined for node j.

f_{ik} = total average traffic in link (i,k), $f_{ik} = \sum_j f_{ik}(j)$.

$t_i(j)$ = total average traffic at node i destined for node j.

$\phi_{ik}(j)$ = fraction of the node flow $t_i(j)$ that is routed through link (i,k).

C_{ik} = capacity of link (i,k).

For later purposes, we shall use a special notation to indicate that node k is a neighbor of i, namely that $(i,k) \in L$. The notation will be $F_i(k) = UP$. The reason for using this notation will become apparent when dealing with topological changes in Chap. 3.

It is now possible to express the law of conservation of flow at each node by various equations. Different equations are used for line switched and message switched networks, the reason being that the controlled quantities are the flows of data for the former and the fractions of the flow for the latter.

2.2.2: Line Switched Network

The flows $f_{ik}(j)$ must satisfy:

$$\sum_{k:F_i(k)=UP} f_{ik}(j) - \sum_{\substack{\ell:F_i(\ell)=UP \\ \ell \neq j}} f_{\ell i}(j) = r_i(j) \quad \text{for all } i,j,i\neq j \tag{2.1}$$

$$f_{ik}(j) \geq 0 \qquad \text{for all } i,j,k, \quad i\neq j \tag{2.2}$$

$$f_{ik} = \sum_j f_{ik}(j) < C_{ik} \qquad \text{for all } (i,k) \in L \tag{2.3}$$

2.2.3: Message Switched Network

The fractions $\phi_{ik}(j)$ must satisfy:

$$t_i(j) - \sum_{\substack{\ell:F_i(\ell)=UP \\ \ell\neq j}} t_\ell(j)\phi_{\ell i}(j) = r_i(j) \qquad (2.4)$$

for all i,j, $i\neq j$

$$\phi_{ik}(j) \geq 0 \;; \quad \sum_{k:F_i(k)=UP} \phi_{ik}(j) = 1 \qquad (2.5)$$

for all i,j,k, $i\neq j$

$$f_{ik} = \sum_j t_i(j)\phi_{ik}(j) < C_{ik} \quad \text{for all } (i,k) \in L \qquad (2.6)$$

In [GALL 77] it is proved that if for each i,j,($i\neq j$) there is a routing path from i to j which means there is a sequence of nodes $i,k,\ell,\ldots,m,j$ such that $\phi_{ik} > 0$, $\phi_{k\ell} > 0$, ..., $\phi_{mj}(j) > 0$ then the set of equations (2.4) has a unique non-negative solution for $t_i(j)$, $i = 1,2,\ldots,N$.

2.3: Delay

Let D_{ik} be the average delay per time unit (seconds), of all traffic sent over link (i,k). Explicitly, D_{ik} is the average delay per unit of traffic (bit, message, packet) multiplied by the amount of traffic per time unit passing through link (i,k). We shall assume that D_{ik} is

only a function of the total flow of traffic f_{ik} transmitted through link (i,k). Some of the consequences of this assumption are given in [GALL 77].

The objective of the algorithms presented here is to minimize the average delay per unit of traffic. However, since the total arrival rate into the network is independent of the routing policy, this objective might be achieved by minimizing the total delay in the network per time unit, which is given by:

$$D_T = \sum_{(i,k)\in L} D_{ik}(f_{ik}) \tag{2.7}$$

The quasi-static algorithms presently described perform this minimization by iteratively changing the routing assignments, while keeping the flow feasible at each iteration.

It should now be pointed out that the algorithms do not require any explicit knowledge of the functions $D_{ik}(\cdot)$ In [KLEI 64] it is shown that under several assumptions the delay in steady state takes the explicit form

$$D_{ik} = \frac{f_{ik}}{C_{ik} - f_{ik}} \tag{2.8a}$$

Another more general well-known form, where propagation and nodal processing times are taken into account, is given by [GERLA 77]:

$$D_{ik} = f_{ik}\left[\frac{1}{C_{ik} - f_{ik}} + \mu(P_{ik} + K_{ik})\right] ; \tag{2.8b}$$

where

$\frac{1}{\mu}$ is the average unit of traffic length; P_{ik} = propagation delay in link (i,k); K_{ik} = nodal processing time in node k.

For the purposes of this work, it is enough to assume only the following reasonable properties of the functions $D_{ik}(\cdot)$:

- D_{ik} is a non-negative continuous increasing function of f_{ik}, with continuous first and second derivatives. (2.9a)

- D_{ik} is convex U (2.9b)

- $\lim_{f_{ik} \to C_{ik}} D_{ik}(f_{ik}) = \infty$ (2.9c)

- $D'_{ik}(f_{ik}) > 0$ for all f_{ik}, where $D'_{ik}(f_{ik}) = \frac{dD_{ik}}{df_{ik}}$ (2.9d)

Observe that the functions in 2.8 indeed have these properties.

2.4: Necessary and Sufficient Conditions for Minimum Delay

In [GALL 77] and [SEG 77a] necessary and sufficient conditions for minimum delay, have been derived for message-switched and line-switched networks, respectively. Here we only indicate their results.

2.4.1: Message Switched Network

If for each $(i,k) \in L$, the functions $D_{ik}(f_{ik})$ have the properties given in (2.9), then a necessary condition for $\phi=\{\phi_{ik}(j)\}$ to minimize D_T over the set of ϕ satisfying (2.6) is that there exists a set of numbers $\{\lambda_i(j)\}$ such that

$$\frac{\partial D_T}{\partial \phi_{ik}(j)} \begin{cases} = \lambda_i(j), & \phi_{ik}(j) > 0 \\ \geq \lambda_i(j), & \phi_{ik}(j) = 0 \end{cases} \tag{2.10}$$

A sufficient condition to minimize D_T is:

$$D'_{ik}(f_{ik}) + \frac{\partial D_T}{\partial r_k(j)} \geq \frac{\partial D_T}{\partial r_i(j)} \qquad \text{for all } i,j,k,\ i\neq j,\ (i,k)\in L \tag{2.11}$$

The last expression (2.11) has been shown to be equivalent to

$$D'_{ik}(f_{ik}) + \frac{\partial D_T}{\partial r_k(j)} - \min_{q:(i,q)\in L} \{D'_{iq}(f_{iq}) + \frac{\partial D_T}{\partial r_q(j)}\} \geq 0 \qquad \text{for all } i,j,k,\ i\neq j,\ (i,k)\in L \tag{2.12}$$

with equality for these i,k,j, such that $\phi_{ik}(j)$ is strictly positive.

The quantity $\partial D_T/\partial r_i(j)$ is the incremental delay caused by a small increment in the input $r_i(j)$ and might be calculated as follows:

$$\frac{\partial D_T}{\partial r_i(j)} = \sum_{k:F_i(k)=\mu P} \phi_{ik}(j)\left[D'_{ik}(f_{ik}) + \frac{\partial D_T}{\partial r_k(j)}\right] \tag{2.13}$$

2.4.2: Line Switched Network

Assume that the set of flows satisfying (2.1), (2.2), (2.3) is nonempty, and let the functions $D_{ik}(f_{ik})$ have the properties given in (2.9) for each $(i,k)\in L$. Then D_T is minimized by the flows $f = \{f_{ik}(j)\}$ if and only if there exists a set of numbers $\lambda=\{\lambda_i(j)\}$ such that

$$D'_{ik}(f_{ik}) + \lambda_k(j) \begin{cases} = \lambda_i(j) & \text{if } f_{ik}(j) > 0 \\ \geq \lambda_i(j) & \text{if } f_{ik}(j) = 0 \end{cases} \tag{2.14}$$

for all i,j,k, $i\neq j$, $(i.k)\in L$, where $\lambda_j(j) = 0$. Observe that if the input flow $r_i(j)$ is increased by an incremental quantity $\delta r_i(j)$ and everything else is held fixed, then the minimum delay will be increased by the incremental quantity $\lambda_i(j)\cdot\delta r_i(j)$. Therefore the coefficients $\{\lambda_i(j)\}$ might be intepretated as marginal delays.

To have a common notation for line-switched and message switched networks, it will be convenient to denote both $\partial D_T/\partial r_i(j)$ in (2.13), (2.14) and $\lambda_i(j)$ in (2.15) by a common notation. Therefore we shall use $d_i(j)$ to denote both the quantity $\partial D_T/\partial r_i(j)$ in the message-switched model and $\lambda_i(j)$ in the line-switched model. Observe that $d_i(j)$ is the marginal node delay, while $D'_{ik}(f_{ik})$ is the marginal link delay.

2.5: The Protocols

2.5.1: Introduction

In this section the routing protocols converging to the minimum delay are first briefly discussed. Then a formal presentation of the protocols is given, which is somewhat different from that in [GALL 77] and [SEG 77a]. Here, the operations required by the algorithms at each node are summarized as a Finite-State-Machine with transitions between states triggered by the arrival of control messages. Control messages are sent between neighbors, queued at the receiving node and processed on a first-come-first-served (FIFO) basis. The processing of a control message consists of temporarily storing it in suitable memory locations, followed by activation of the Finite-State-Machine, which takes the necessary actions and performs the appropriate state transitions. Some variables are used as conditions for the execution of transitions, and their values might be changed by the transitions.

For readers familiar with Gallager's algorithm, we note that we introduce here a slight modification. In [GALL 77], the updating of the quantities $d_i(j)=\partial D_T/\partial r_i(j)$ is performed while the protocol propagates from each destination upstream in the network, while the timing of the actual rerouting is left arbitrary. For later purposes, related to topological changes, it will be convenient that we introduce already at this point a certain sequencing for rerouting. Specifically, the update of $\{d_i(j)\}$ will be performed as before while the protocol propagates upstream, but now we will also have a propagation of the protocol in the downstream direction, during which the nodes will actually change their routing.

2.5.2: Informal Description of the Protocol

Considering the optimality conditions for the two models, the general structure of the algorithms should be clear. A node i will have to increase traffic destined for node j on links (i,k) with small marginal delay $D'_{ik}(f_{ik}) + d_k(j)$ and to decrease traffic on those with large marginal delay.

Obviously, in addition to the quantities $d_k(j)$ that it receives from neighbors, each node i will need the marginal delay $D'_{ik}(f_{ik})$ over each of its outgoing links. $D'_{ik}(f_{ik})$ can be obtained by node i by estimating f_{ik} and using appropriate formulas for $D'_{ik}(f_{ik})$. However, each formula involves many assumptions, so node i should preferably estimate $D'_{ik}(f_{ik})$ directly. Such estimation procedures have been developed in [SEG 77b], and from now on we assume that each node i continuously estimates or calculates the marginal delay $D'_{ik}(f_{ik})$ over each of its outgoing links (i,k).

We should also note here that the optimality conditions clearly show that different destinations are not related, so that the protocols may evolve independently from one destination to another. That is why the protocols are presented for a given fixed destination j which is denoted by SINK from now on.

Before proceeding, the following definitions are needed.

Son: For message switched network: All neighbors k of node i (namely all nodes k s.t. $F_i(k)$=UP) with $\phi_{ik}(SINK) > 0$ are called its sons (see Fig. 2).

For line switched network: All neighbors k of node i (namely all nodes k s.t. $F_i(k)$ =UP) with $f_{ik}(SINK) > 0$ are called its sons. In case that $f_{ik}(SINK) = 0$ for all neighbors k, then node i has exactly one son; this is its preferred neighbor to which it would send any flow destined for SINK if such flow comes in.

Father: Node k is a father of node i if node i is a son of node k.

Downstream node: Node ℓ_1 is downstream from node ℓ_q if there is a set of nodes $\ell_2,\ell_3,\ldots,\ell_{q-1}$ such that ℓ_i is a son of ℓ_{i+1} for $i = 1,2,\ldots,(q-1)$. (see Fig. 2)

Upstream node: Node ℓ_1 is upstream from node ℓ_q if node ℓ_q is downstream from node ℓ_1. (see Fig. 2).

Loop: A set of nodes $\ell_1,\ell_2,\ldots,\ell_q,\ell_1$ form a loop if node ℓ_1 is both upstream and downstream from node ℓ_q. (see Fig. 2).

We are now ready to describe the algorithms. Each node i in the network has, for each neighbor k, memory locations called $N_i(k)$, $D_i(k)$, $B_i(k)$ and $R_i(k)$. $N_i(k)$ denotes a flag which can take the value RCVD to mean that a control message was received at i from k during the current iteration, or the value NIL otherwise. $D_i(k)$ and $B_i(k)$ are kept for storing of control messages received at i from k. $R_i(k)$ denotes an indicator which can take the value SON to mean that node k is a son

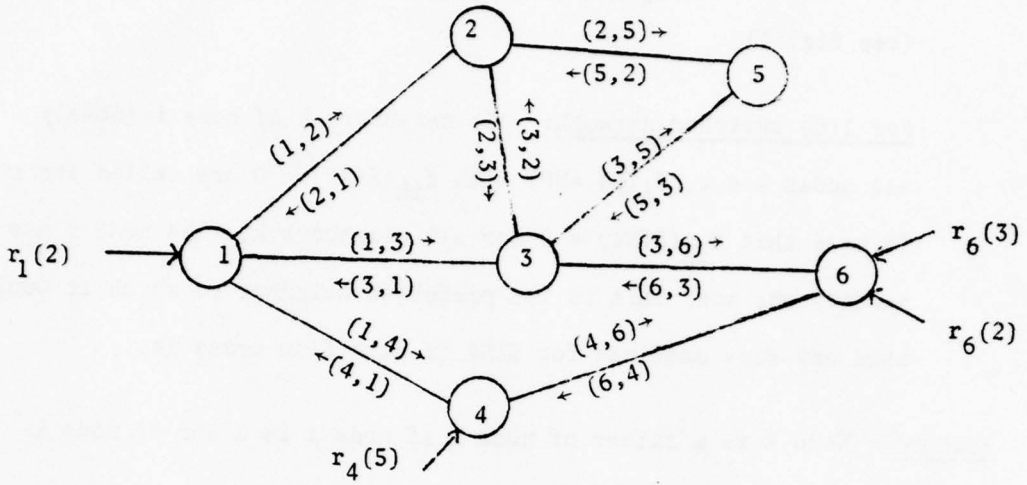


Fig. 1: Nodes, links and inputs in a network.

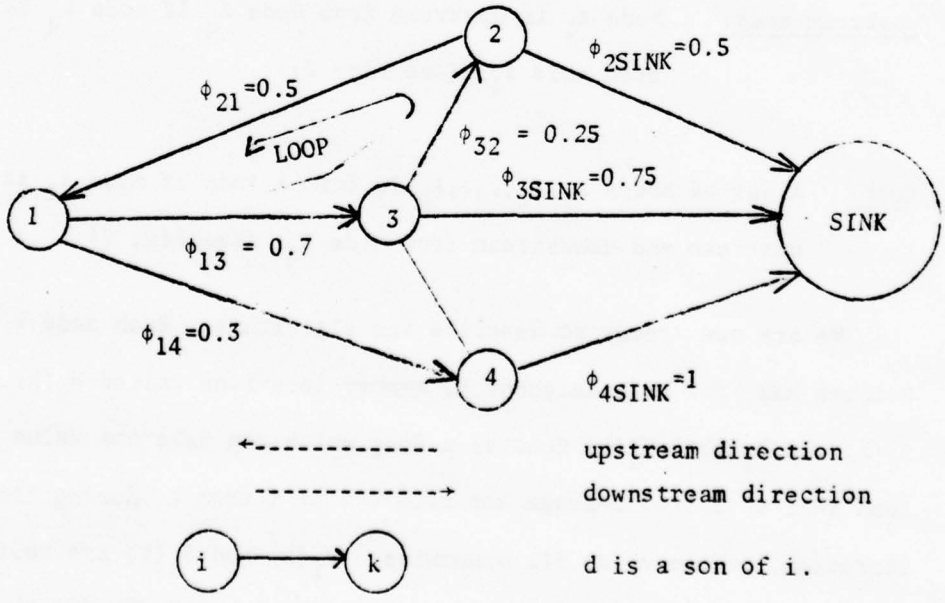


Fig. 2: Routing, sons and loops.

of node i, or the value NIL otherwise.

In addition, node i keeps its estimated marginal delay d_i to the SINK and its routing variables ϕ_{ik} (in message-switching) or f_{ik} (in line-switching), for each neighbor k.

During the activity of the protocols , control messages are sent between neighbors. These messages contain the estimated marginal delay d_ℓ of the sender ℓ, to the SINK. The control messages are processed on a FIFO basis at the receiving node. At first, the processor at the receiving node, i say, identifies the sender, ℓ say, of the received message, and rises its $N_i(\ell)$ flag, i.e. sets $N_i(\ell)$ = RCVD, then adds to the received d_ℓ the current estimated marginal delay $D'_{i\ell}$ on link (i,ℓ) and stores the sum in $D_i(\ell)$.

Suppose now that there is a procedure which keeps the network loop-free at all times. Each iteration of the protocols is started when the SINK enters state named S2, and sends a message with $d_{SINK} = 0$ to all its neighbors. Let us now restrict ourselves to an arbitrary node i in the network and describe its activities during an iteration of the protocols. The Finite-State-Machine for each node in the network is given in Fig. 3. Generally speaking, a node i enters the state S2 when it has received control messages from <u>all its sons</u>. At this time it also updates its estimated marginal delay d_i and sends the updated d_i to all neighbors except its sons. The return to state S1 is performed when the node has received control messages from <u>all its neighbors</u>. At this time the estimated marginal delay d_i is sent to the sons and routing changes are performed at node i.

Before proceeding to explain the updating of d_i and of the routing variables we describe the procedure of keeping the network loop-free. The concept of blocking introduced in [GALL 77] is needed here. Briefly, if the flow from node i over link (i,k) (destined for SINK) is strictly positive and $d_k \geq d_i$, then there is danger of producing a loop in the next iteration of the algorithms. To avoid this, if because of the constraints on the step-size involved in the algorithms, node i is not sure that it can reroute all the flow on (i,k) in one step, then it declares itself blocked, and so do all nodes upstream from it. It is shown in [GALL 77] and [SEG 77a] that loops are not generated in the network if the following rule is kept: The flow to a blocked node which is not a son is not allowed to be increased from zero.

Updating of d_i is done when transition from state S1 to state S2 occurs. We denote this transition by T12. For the message switched network d_i is calculated using formula (2.13). For the line switched network, d_i is calculated as the minimum of all $D_i(k)$ received by i up to this point from all sons and other nonblocked neighbors. In addition, when entering S2, node i updates its blocking status so that any potential loop will be prevented. Then it sends (the updated) d_i and its blocking status to all neighbors except sons.

While entering state S1, from state S2, namely when transition T21 occurs, node i reroutes the flow (destined for SINK) in a particular way, so that both convergence of the protocols to the minimum delay routing and the loop-freedom property are insured. This is done by choosing a preferred son, through which the flow might be increased. Then the routing variables are changed, d_i is sent only to sons and then the list of sons is updated.

Notice that according to the protocols, the updating of the estimated marginal delays $\{d_i\}$ propagates from the SINK upstream and the rerouting proper propagates downstream towards the SINK. Clearly, this procedure is deadlock free if and only if there are no loops in the network. We see therefore that maintaining loop-freedom in the network at all times is essential to provide a natural sequencing in the network, in addition to saving resources.

Notice also that transition of SINK from state S2 to state S1 (remember the SINK enters S2 when starting an iteration) means completion of the whole iteration by the entire network. The SINK is then allowed to start a new iteration anytime, provided it is in state S1.

2.5.3: Formal Description of the Protocols

We are now going to display the formal protocols. First, we define the variables used by the algorithms at node i, then the algorithms performed by each node are displayed. At last, the algorithms performed by SINK are presented.

Since the algorithms for the two introduced models are very similar, we describe them simultanously and when applicable, indicate the differences.

Definition of variables:

The values a variable can take appear in parentheses.

i = node under consideration;

ℓ = the ℓ-th neighbor of node i: (1,2,...,N);

η = a parameter: (see Theorem 2.2);

d_i = estimated marginal delay of node i from SINK:(1,2,...);

b_i = blocking status of node i:(0,1); 0 means not blocked; 1 means blocked;

MSG (d,b,ℓ) = control message received by i from neighbor ℓ:(d=d_ℓ, b=b_ℓ);

$D'_{i\ell}$ = estimated (or calculated) marginal delay on link (i,ℓ):(1,2,...);

$d_i(\ell)$= last number d received at i from neighbor ℓ:(0,1,...);

$N_i(\ell)$= flag:(NIL, RCVD); RCVD means a message has been received at i from neighbor ℓ during the current iteration;

$D_i(\ell)$= $d_i(\ell) + D'_{i\ell}$:(1,2,...);

$B_i(\ell)$= blocking status of neighbor ℓ as known at i:(0,1); 0 means not blocked; 1 means blocked;

$R_i(\ell)$= status of neighbor ℓ:(NIL,SON); SON means node ℓ is a son of i.

In the formal description of the actions done by node i, we need the following sets of neighbors:

C_i = set of all nodes k such that $R_i(k)$ = SON and all other nodes k with $F_i(k)$ = UP, $N_i(k)$ = RCVD and $B_i(k)$ = 0.

A_i = set of all nodes k such that $R_i(k)$ = SON and all other nodes with $F_i(k)$ = UP and $B_i(k)$ = 0.

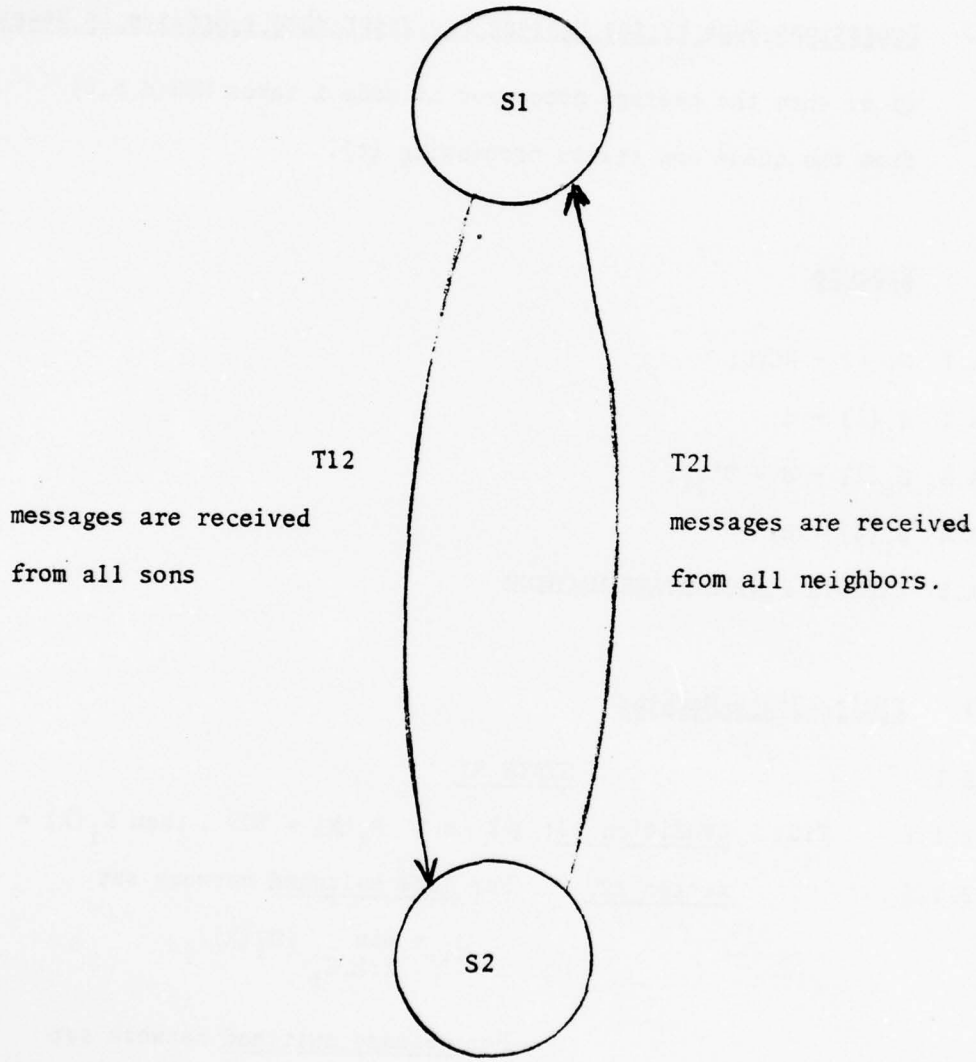


Fig. 3: Finite-State-Machine for an arbitrary node (Basic algorithms).

The Algorithms

(for each node i except SINK).

A. Operations Done by the Message Processor when a Message is Received

(i.e. when the message processor at node i takes MSG(d,b,ℓ) from the queue and starts processing it).

Execute

A.1 $N_i(\ell) \leftarrow$ RCVD;

A.2 $d_i(\ell) \leftarrow d$;

A.3 $D_i(\ell) \leftarrow d + D'_{i\ell}$;

A.4 $B_i(\ell) \leftarrow b$;

A.5 EXECUTE FINITE-STATE-MACHINE.

B. Finite-State-Machine

B.1 STATE S1

B.1.1 T12: Condition 12: $\nexists k$ s.t. $R_i(k) = SON$, then $N_i(k) = RCVD$;

B.1.2 Action 12: For line switched network set

$$d_i \leftarrow \min_{k:k\in C_i} \{D_i(k)\}; \tag{2.15}$$

For message switched network set

$$d_i \leftarrow \sum_{k:R_i(k)=SON} \phi_{ik} \cdot D_i(k); \tag{2.16}$$

B.1.3 Check of status: If for any k s.t. $R_i(k) = SON$ then $\{B_i(k) = 1\}$ or

{for line switched network

$$d_i(k) \geq d_i \;\&\; \eta[D_i(k) - d_i] < f_{ik}; \quad (2.17)$$

for message switched network

$$d_i(k) \geq d_i \;\&\; \eta[D_i(k) - d_i]/t_i < \phi_{ik};\} \quad (2.18)$$

then set $b_i \leftarrow 1$; otherwise set $b_i \leftarrow 0$;

B.1.4 $\forall k$ s.t. $F_i(k) = UP$ and $R_i(k) \neq SON$, send (d_i, b_i, i);

B.2 STATE S2

B.2.1 T21: Condition 21: $\forall k$ s.t. $F_i(k) = UP$, then $N_i(k) = RCVD$;

B.2.2 Action 21: Rerouting:

Calculate $\alpha = \min_{k:k\epsilon A_i} \{D_i(k)\}$; (2.19)

B.2.3 let k_o be any neighbor that achieves the minimum in (2.19);

B.2.4 For line switched network:

If there is any node q s.t. $F_i(q)$ with $f_{iq} > 0$, then for all neighbors $k\epsilon A_i$ do:

B.2.4.1 $a_{ik} = D_i(k) - \alpha$; (2.20)

B.2.4.2 cancel all outgoing links corresponding to incoming links, that have been cancelled by fathers. Let f'_{ik} be the remaining flows.

B.2.4.3 $\Delta_{ik} = \min\{f'_{ik}, \eta \cdot a_{ik}\}$; (2.21)

B.2.4.4 SET NEW FLOW ($\forall k$ s.t. $F_i(k) = UP$)

$$f_{ik}^{new} = \begin{cases} 0 & k \notin A_i \\ f'_{ik} - \Delta_{ik} & k \in A_i, k \neq k_o \\ f'_{ik} + \sum_{\substack{k \in A_i \\ k \neq k_o}} \Delta_{ik} + \text{any new flow}, & k = k_o \end{cases} \quad (2.22)$$

B.2.4.5 If $f_{ik} = 0$ $\forall k$ s.t. $F_i(k) = UP$, then any new flow is routed through k_o;

B.2.5 For message switched network:

if $t_i > 0$, then for all neighbors $k \in A_i$ do:

B.2.5.1 $a_{ik} = D_i(k) - \alpha$; (2.23)

B.2.5.2 $\Delta_{ik} = \min\{\phi_{ik}, \eta a_{ik}/t_i\}$; (2.24)

B.2.5.3 SET NEW FLOW: ($\forall k$ s.t. $F_i(k) = UP$)

$$\phi_{ik}^{new} = \begin{cases} 0 & k \notin A_i \\ \phi_{ik} - \Delta_{ik} & k \in A_i, k \neq k_o \\ \phi_{ik} + \sum_{\substack{k \in A_i \\ k \neq k_o}} \Delta_{ik} & k = k_o \end{cases} \quad (2.25)$$

B.2.5.4 if $t_i = 0$, then set $\phi_{ik_o} = 1$ and $\forall k \neq k_o$ set $\phi_{ik} = 0$;

B.2.6 $\forall k$ s.t. $R_i(k) = SON$, send (d_i, b_i, i);

B.2.7 $\forall k$ s.t. $F_i(k) = UP$, set $R_i(k) = NIL$;

Set $R_i(k_o) = SON$; Set $R_i(k) = SON$ $\forall k$ s.t. $F_i(k) = UP$, $k \neq k_o$ and $f_{ik}^{new} > 0$ (for line switched network) or $\phi_{ik}^{new} > 0$ (for message switched network);

B.2.8 $\forall k$ s.t. $F_i(k) = UP$, set $N_i(k) = NIL$;

This completes the description of the algorithms for all nodes in the network except the SINK. The SINK performs the same operations as all other nodes and in addition, it can start a new iteration at any time, provided it is in state S1, by going into S2 and transmitting MSG(d=0,b=0,SINK) to all nodes k s.t. $F_{SINK}(k) = UP$.

Finite-State-Machine for SINK

STATE S2

T21: Condition 21: $\forall k$ s.t. $F_{SINK}(k) = UP$, then $N_{SINK}(k)$=RCVD.

Action 21: $\forall k$ s.t. $F_{SINK}(k) = UP$, set $N_{SINK}(k) = NIL$.

2.5.4: Properties of the Protocols

The most important properties of the two quasi-static protocols described before are:

(1) Distributed computation is used.

(2) Loop-freedom (for each destination) is maintained in the network at all times.

(3) Convergence to the minimum delay under certain conditions.

The properties are rigorously stated in the following theorems, whose proofs appear in [GALL 77], [SEG 77a].

Theorem 2.1 (Loop-Freedom)

At all times, the flows to each destination are loop-free.

Theorem 2.2: (Convergence)

Let the input traffic into the network be stationary, and let the topology of the network be fixed. Then under assumptions (2.9), there is a sufficiently small value of the parameter η such that D_T converges to the value of the minimum average delay over all routing assignments, for any initial flow.

2.5.5: Initialization of the Protocols

Obviously, the protocols must be started with some loop-free flow. The following starting rule is suggested in [SEG 77a]: When the network starts operating, each node chooses its son to be the first node from which it receives a control message. Thus an upstream relation is established and the algorithms can continue as usual. This topic is further discussed in the next chapter when addition of links to the network is taken into account.

CHAPTER 3

ROUTING PROTOCOLS FOR NETWORKS WITH CHANGING TOPOLOGY

3.1: Introduction

The protocols described in Chapter 2,referred to from now on as the basic protocols, can operate smoothly only when no topological changes occur in the network and in that case, they gradually adapt to changes in the traffic requirements. However, since nodes and communication links occasionally fail and recover in any practical network, the basic protocols should be expanded to handle arbitrary topological changes, while preserving the main properties of the basic protocols. The protocols presented in this chapter are designed to do so, independently of the number, timing and location of those topological changes. These protocols are a natural extension of the protocol of [SEG 77c] where a single optimal route was maintained from each node to the destination. Essentially, the extra feature provided by the pfesent protocols compared to the protocol of [SEG 77c] is to indicate the exact amount of flow splitting so that optimal average delay is obtained in the network in steady-state. As such, our resulting protocols, to be presented in the subsequent sections, have all of the following properties:

(1) Distributed computation is used.

(2) Loop-freedom for routes for each destination is maintained at all times.

(3) Recovery of the network in finite time from arbitrary number, timing and location of topological changes.

(4) If the traffic is stationary and the topology fixed for long enough time, the network is brought in steady-state to the minimum delay routing.

In general, the description of these protocols follows the same pattern as for the basic protocols. The main basic changes are that the finite-state-machine contains more states and the control messages contain more information. The entire extension is given in the subsequent sections.

To analyse the protocols and validate their properties and correctness, a technique introduced in [SEG 77c] is used. According to this technique, a special type of induction is used, that allows to prove global properties while essentially looking at local events. The main proofs are given in the appendices.

The extension of the basic protocols is exactly the same for both message and line switching. Consequently, in the informal description we do not distinguish between the two models, and return to do so only in the formal description. In the last section of this chapter all properties of the resulting protocols are formally stated in a series of theorems, whose proofs appear in the appendices.

3.2: Informal Description of the Protocol

3.2.1: Introduction

The operations to be performed by the algorithm at each node in "normal" conditions, namely when no topological changes occur in the network, have been described in the basic protocols. Here we first give the general additions to the protocol and then the details are provided, first for link failures and then for links becoming operational. We do

not pay special attention to topological changes caused by nodes, since such changes might be perceived as the change of status of all links connected to those nodes.

This protocol is still operating independently for each destination and as before, we present it for a given fixed destination called SINK.

3.2.2: General Additions

For later purposes, there is need to number the interations of the protocol with nondecreasing numbers as described below. Each node i will have a node counter number n_i which denotes the iteration number currently handled by this node. All control messages transmitted by i will carry n_i in addition to d_i and b_i, namely they will be of the form MSG(m,d,b,i) with $m=n_i$, $d=d_i$ and $b=b_i$. When a MSG(m,d,b,ℓ) is received by node i on link (i,ℓ), then d and b are stored in $D_i(\ell)$ and $B_i(\ell)$ respectively, as dictated by the basic algorithms, and in addition there is also need to remember the value of m. This value can be saved in $N_i(\ell)$, which can now take the values NIL, 0,1,2,...; instead of NIL and RCVD as for the basic algorithms. For simplicity, the parameter ℓ in MSG(m,d,b,ℓ) is suppressed from now on.

Generally speaking, after having received MSG(m,d,b) with a given counter number m from all its current sons, node i updates its counter number n_i to m and effects transition T12 in the same way as for the basic algorithms. Transition T21 is performed when the node receives MSG(m,d,b) with counter number m from all its current neighbors.

In our extended protocols the SINK starts consecutive iterations with nondecreasing counter number. If the SINK starts an iteration with counter number $m = n_{SINK}$ and completes it before starting a new iteration, we say that there has been a proper completion. We denote the time of proper completion of an iteration with number m by PC(m). In this case, the SINK is allowed to start a new iteration with the same counter number.

To handle topological changes, there are situations that the SINK must increase the iteration counter number. The protocol allows it to do so at any time, whether the previous iteration was completed or not. (Notice that in any case the values of n_{SINK} are nondecreasing with time). As proved later, if a new iteration is started while increasing n_{SINK}, it will eventually cover all previous iterations.

There are several possible ways to insure that the SINK increases its n_{SINK} (and starts an iteration with this number) finite time after need arises. These possibilities are described in detail in Chapter 4, but for the purposes of this chapter it is enough to assume that such an algorithm indeed exists. The formal assumption is given in Assumption A in Section 3.4.3.

3.2.3: Handling Failures of Links

We now turn to describe the algorithm for a node i that discovers a failure on one of its incident links. We assume here (see formal assumptions 7,9, in Section 3.3.2) that whenever link (i,ℓ) fails then link (ℓ,i) fails at the same time, but the nodes i and ℓ may discover the failure

at different instants. However, if i discovers a failure on(i,ℓ), it cannot bring the link up before ℓ discovers the failure too.

There are three typical situations to be distinguished. First, the case when the node has only one son and discovers a failure on the link to this son. Second, the situation when the node has more than one son, and the failure is discovered on the link to one of its sons. Third, the case when the failure occurs on a link to a neighbor that is not a son. In all these cases, the first action node i takes when discovering a failure on link (i,ℓ) is to set $F_i(\ell)$ = FAIL, thus indicating that node ℓ is not a neighbor any longer. Now, the role of $F_i(\ell)$ becomes apparent. $F_i(\ell)$ indicates the status of link (i,ℓ) as seen from node i. $F_i(\ell)$ = UP means that the link (i,ℓ) is under normal operation, namely that ℓ is a neighbor of i. $F_i(\ell)$ = DOWN means that the link (i,ℓ) is unoperational. $F_i(\ell)$ can also take the value READY whose use will become apparent when dealing with links becoming operational.

Single Son

If node i has only one son, ℓ say, and link (i,ℓ) fails, then node i loses its only route to the SINK. In addition, some nodes upstream from node i lose one or more of their routes to the SINK. However, all those upstream nodes are unaware of this fact at the time the failure occurs. For instance (see Fig. 4), if link (6,1) fails then nodes 6,8,5,9 lose all their routes to the SINK and nodes 4,7 and 10 lose one of their routes to the SINK. Furthermore, if an

iteration is started by the SINK, node 6 will never be able to receive a control message from node 1, and therefore, node 6, as well as nodes 4,5,7,8,9 and 10, will never be able to perform T12. The extentions to the basic algorithms provided here are designed to allow the network to recover from this situation, namely to provide alternate routes to nodes that have lost their only route to the destination and to allow the other affected nodes to continue their normal operation. This and the next subsections indicate these actions.

Two actions must be taken by the extended protocol. First, to inform the nodes upstream from node i not to wait for control messages from their sons that are on the failed routes, and also to notify them that the routes do not exist any longer (e.g. in Fig. 4 node 8 should be informed that the path 8,6,1, SINK does not exist any longer). Second, to allow node i (and possibly its upstream nodes that lost all their routes) to choose a new son whenever control messages of <u>new</u> iterations will be received. This features are described presently.

Whenever a node i discovers a failure on link (i,ℓ), where ℓ is its only son, it sets $R_i(\ell)$ = NIL and $d_i = \infty$ to mean that ℓ is no longer its son and that its marginal delay to the SINK has become infinite. Then node i generates a special control message, $MSG(n_i, d=\infty, b_i)$ which is sent to all neighbors of i except ℓ; if a node k receives such a message from its <u>only son</u>, q say, then it performs similar operations, namely it sets $R_k(q)$ = NIL, $d_k = \infty$ and sends $MSG(n_k, d=\infty, b_k)$ to all its neighbors except node q; if a node receives such a message from a neighbor that is not a son, it stores it but no other action is taken; the case when the node has more than one son and such a message is received from some son

is discussed in the next subsection. When a node i establishes $R_i(\ell)$ = NIL for its single son and $d_i = \infty$, it also enters state S3.

A node that enters state S3 must select a new son, thus establishing a new route to the SINK. This procedure is the second part of the recovery and is called reattachment. The reattachment takes place if one of the following two situations occurs. One possibility is when a node i in state S3, (and hence with no sons), receives a control message MSG(m,d≠∞, b) from ℓ say, with $m > n_i$. Then node i knows that this message was generated by an iteration started after the failure that caused its entrance to state S3. A second possibility is that such a message has already been received at i from ℓ at the time i enters state S3. The reattachment consists of setting $R_i(\ell)$ = SON, going to state S2 and effecting the same operations as in T12. This, together with other procedures to be presented, guarantees that if any number of failures occur, and if the SINK really starts an iteration that cover all these failures (as we assumed) i.e. an iteration with counter number that was not the node number of any of the nodes detected the failures while detection, then each node physically connected to the SINK will eventually have at least one route to the SINK. Furthermore, no loops are generated by the reattachment procedure, a property that is stated in Theorem 3.1 and proved in the appendices..

More than one Son

If node i has more than one son and a failure is discovered by node i on link (i,ℓ) connecting it to one of its sons, ℓ say, or if node i receives MSG(m,d=∞,b) from ℓ, then node i knows that its route to the SINK passing thourgh ℓ has been destroyed (e.g. failure on link (4,5)

in Fig. 4 detected by node 4). Naturally, $R_i(\ell)$ must be set to NIL to indicate that ℓ is not a son any longer, but no transition should be performed and no special action is to be taken, since node i still has other sons. This is accomplished by introducing C1 (and later also $\tilde{C2}$) into the Finite-State-Machine, in which the only action taken is to set $R_i(\ell)$ = NIL, while staying in the same state. However, if node i is in state S2, it is necessary to prevent T21 from happening at this node for the current iteration, the reason being that this will prevent nodes from updating their routes based upon information which is invalidated by the failure. We rather explain the last sentence. While entering state S2 node i calculates its marginal delay d_i to the SINK and determines its blocking status b_i. Suppose that while being in S2, the link to the node that the calculation of d_i was based upon, fails. Then there is a danger that its blocking status b_i is incorrect. Therefore, if T21 will be performed, a loop might be generated - a situation we must avoid. Another important reason is that prevention of T21 from happening will also preclude proper completion from happening. Thus, proper completion will now indicate to the SINK that the iteration was completed without failures interfering with the process. Prevention of T21 is accomplished by introducing an additional state, $\tilde{S2}$, into which a node enters if it has more than one son and either detects a failure on a link connecting it to one of its sons, or receives MSG(m,d=∞,b) from it, while being in state S2. A node i will leave $\tilde{S2}$ whenever it receives new control messages from <u>all its sons</u>. To be more specific, node i leaves $\tilde{S2}$ by either going to state S3 in case a failure is sensed on its single route to the SINK (as we have already described), or going to state S2 when receiving MSG(m,d≠∞,b) of a new iteration, i.e. $m > n_i$,

from all its sons, and in this case it effects the same operations as in T12.

The intention of the part of the algorithm described in the last two subsections is, to enable upstream propagation of the knowledge of a failure occurrence. All nodes that are upstream from the failure are informed that they cannot send any flow through the failed routes. In addition, neighbors of the nodes that have lost all their routes to the SINK, are informed not to choose these nodes as their sons.

Clearly, in addition to all the above operations, each node that has lost one of its sons should stop the flow to that son, transmit it through its remaining sons, if it still has any, and modify its routing variables correspondingly. The question of how should the node distribute the flow between its remaining sons is immaterial for the purposes of our work. This is because a failure usually causes dramatic changes in the routing variables, thus in the total delay, so the exact distribution is unimportant in our quasi-static algorithm that allows only fine changes. We may assume that it is done in some way that insures that the capacity constraints are not violated. Another open question is what should the node do with the flow if it has no route to the destination. In this case we may assume that it stores the flow until it establishes a new route, if it can, otherwise, it rejects it.

Failure of a neighbor that is not a son

Up to now we have described the algorithm for a node i discovering a failure on a link connecting it to one of its sons. If failure occurs on a link to a neighbor which is not a son, (e.g. link (6,3) in Fig. 4),

then no route is disrupted, so no special action is needed. However, for reasons explained in the previous subsection, if the failed link is connected to a node in state S2, it is convenient here also to prevent T21 from happening at this node for this iteration. Consequently, a node enters $\tilde{S2}$ whenever a link to nonson fails while the node is in state S2. The procedure of leaving $\tilde{S2}$ has already been described.

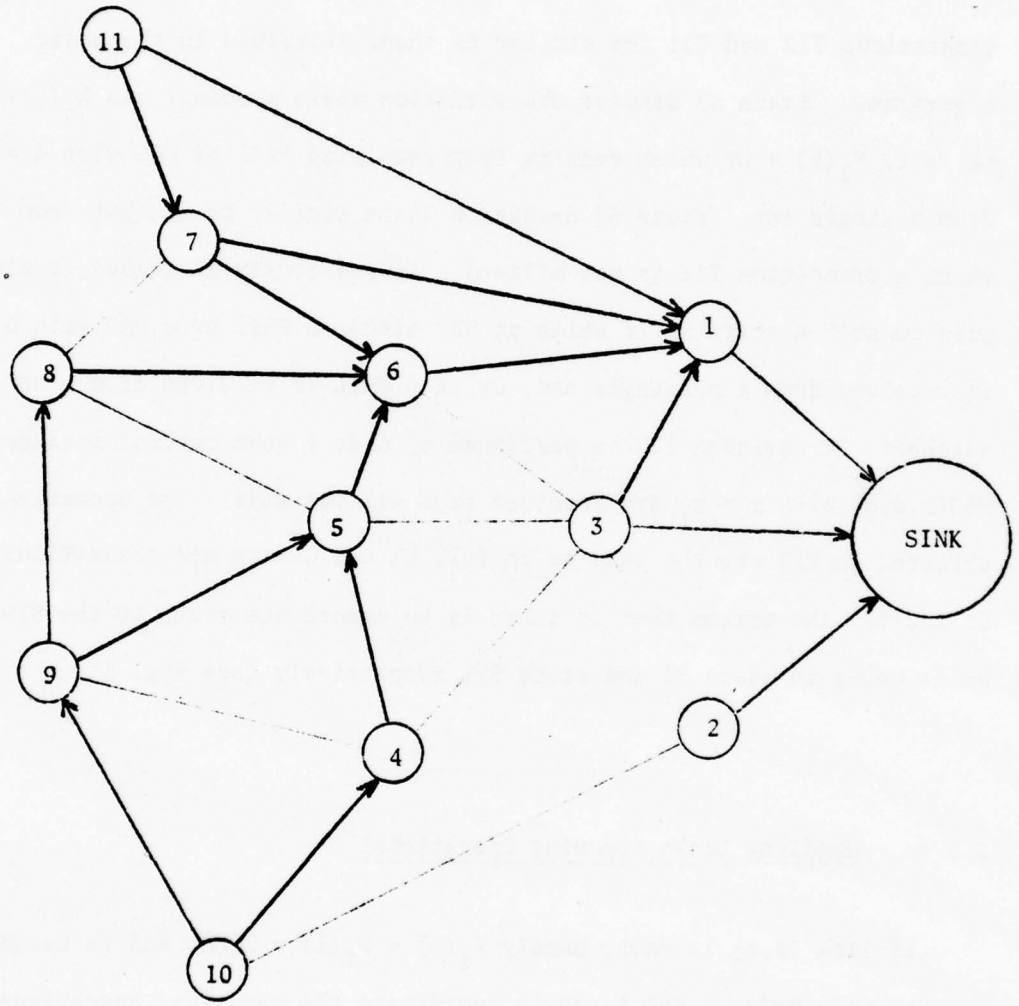
The protocol as described up to now is implemented by the algorithm in the formal description given in Section 3.3.3 if ignoring steps A.2, A.2.1 - A.2.4, A.3.1, B.1.7, B.2.8, B.8.8. These steps relate mainly to links becoming operational and will be discussed in the subsequent section. The notations used here are similar to those in chapter 2. A summary of these notations is given in Subsection 3.3.1. There, the variables used by the algorithm performed by an arbitrary node i as its part of the protocol, are given. $F_i(\ell)$ denotes the status of link (i,ℓ) as considered by node i, namely $F_i(\ell)$ = UP if ℓ is considered operational and $F_i(\ell)$ = DOWN if ℓ is considered unoperational. $F_i(\ell)$ can also take the value READY, whose use will become clear when dealing with the problem of links becoming operational. At that time the role of $Z_i(\ell)$ will also become apparent. The variable mx_i stores the value of the largest counter number m of all messages MSG(m,d,b) received by node i from all its neighbors.CT plays a role of a flag indicating the number of transitions that have already been performed by the Finite-State-Machine, triggered by the current message, 0 will mean zero transitions, 1 will mean one or more transitions. The rest of the variables and their use were already described. The link (local) protocols controlling the operations of the links connected to node i may relay to the algorithm performed by node i three types of messages. MSG denotes an

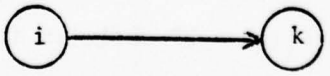
updating message, FAIL(ℓ) denotes the detection of the failure of link (i,ℓ). The remaining message WAKE(ℓ) is described later.

To give here a short summary, remember that states S1 and S2 and transitions T12 and T21 are similar to those described in the basic algorithms. State S3 denotes the situation where a node i has $R_i(k)$=NIL $\forall k$ s.t. $F_i(k)$ = UP which results from receiving FAIL or MSG with d = ∞ from a single son. State $\tilde{S2}$ denotes a state similar to S2, but from which a transition T21 is not allowed. As previously described, a node goes to such a state $\tilde{S2}$ if while at S2, either a FAIL or a MSG with d = ∞ is received from a nonsingle son, or if a FAIL is received from a nonson neighbor. Transition T22 is performed by node i when control messages MSG(m,d,b) with $m > n_i$ are received from all its sons. The operations effected in T22 are the same as in T12. C1 and $\tilde{C2}$ are not transitions. In C1 and $\tilde{C2}$, the action that is taken is to cancel one route to the SINK while being in state S1 and state $\tilde{S2}$, respectively (see Fig. 5).

3.2.4: Handling Links Becoming Operational

If link (i,ℓ) is down, namely $F_i(\ell) = F_\ell(i)$ = DOWN, and it becomes operational, nodes i and ℓ should coordinate the necessary operations to bring the link up. Otherwise, a deadlock can occur. For instance, suppose node i sets $F_i(\ell)$ = UP while at state S2 and node ℓ sets $F_\ell(i)$ = UP after performing T21 of the same iteration. In this case, node i will not perform T21 until receiving a control message from node ℓ, and such a message will not be sent because ℓ has already completed this iteration, i.e. deadlock.





denotes k is a son of i.

Fig. 4: Network example (with loop-free routing).

The coordination is achieved by having both nodes bring the link up just before starting to perform their part of the same new iteration. This is done as follows: When nodes i and ℓ sense that link (i,ℓ) becomes operational, they compare their node counter numbers, n_i and n_ℓ, via their link (local) protocol, and decide to bring up the link when starting to process the first iteration with a number strictly higher than $\max(n_i,n_\ell)$. This fact must be remembered at the nodes and it is done be setting the memory locations, $Z_i(\ell)$ at i and $Z_\ell(i)$ at ℓ, to $\max\{n_i,n_\ell\}$. Clearly, $\{Z_i(k)\}$ are memory locations kept at i for each possible neighbor k of i. In addition to the above operations, nodes i and k also set $F_i(\ell)$ and $F_\ell(i)$ to READY, and $N_i(\ell)$ and $N_\ell(i)$ to NIL. In order to bring the link up, there is need that the SINK will start an iteration with n_{SINK} larger than $Z_i(\ell)$ (and $Z_\ell(i)$). By the assumption we made in Section 3.2.2, such an iteration is indeed started in finite time. (See also Assumption A in Sec. 3.4.3).

The execution of the first step of the coordination at node i is triggered by a special control message - WAKE(ℓ) given by the link (local) protocol to the algorithm at noe i (and similarly WAKE(ℓ) is delivered to the algorithm at node ℓ). The actions performed by the algorithm when receiving such a message are described in A.2, A.2.1 - A.2.4 in the formal description in Section 3.3.3. This synchronization assumes that the execution of WAKE(ℓ) and WAKE(i) are simultaneously started at nodes i and ℓ respectively, in order to guarantee that $Z_i(\ell) = Z_\ell(i)$. However, it may happen that a failure occurs again on the link and one of the nodes succeeds to complete the synchronization while the other node does not. The protocol allows for such a situation and only requires that the link protocol ends the synchroniation (successfully or unsuccessfully) within finite time. If the synchronization is unsuccessful, no action is taken by

the node, and the link will remain DOWN from this node's point of view. Section 3.3.2 gives a more formal and complete list of the requirements that the link (local) protocol should satisfy.

The link (i,ℓ) is finally brought up by node i, namely, $F_i(\ell)$ is set from READY to UP, when node i receives MSG from link (i,ℓ), or when the node counter number n_i becomes larger than $Z_i(\ell)$.

3.2.5: The Algorithm for the SINK

The algorithm for the SINK is similar to that for an arbitrary node i, except that the SINK does not need to keep the following variables:

- $R_{SINK}(\ell)$ (which is not defined for the SINK, since it has no sons).
- d_{SINK} (which is 0 by definition for the SINK).
- b_{SINK} (which is 0 by definition for the SINK, since it has no sons).
- $D_{SINK}(\ell)$ (which is only needed to update d_{SINK} and $R_{SINK}(\ell)$).
- mx_{SINK} (n_{SINK} is always the largest update counter number).
- $Z_{SINK}(\ell)$ (during WAKE synchronization $Z_{SINK}(\ell)$ is always set to $n_{SINK} = \max\{n_{SINK}, n_\ell\}$.)

In addition, the SINK can start a new iteration at any time, by going to state S2 and sending MSG(n_{SINK},d=0,b=0) to all its current neighbors, provided that the last iteration was properly completed. Moreover, when necessary, the SINK increments its n_{SINK} and starts a new iteration as

described above, even if the last iteration was not properly completed. We have seen that there is need to increment n_{SINK} and start a new iteration whenever a topological change occurs in the network. The exact details of how this can be actually done in a distributed way are provided in the next chapter.

In the algorithm for the SINK, states S1 and S2 are similar to the corresponding states of the algorithm for an arbitrary node i. However, for the SINK, S1 means that the last iteration was properly completed, and S2 means that the last iteration is not yet completed. T12 and T22 represent the starting of a new iteration and T21 represents proper completion (see Fig. 6). For the SINK there is no need for states equivalent to S3 and $\tilde{S2}$ of the algorithm for an arbitrary node because whenever the SINK detects a topological change it starts immediately a new iteration while incrementing n_{SINK}.

3.2.6: Initialization of the Protocol

A node i comes into operation in state S3, with node counter number $n_i = 0$, and $R_i(k) = NIL$, $F_i(k) = DOWN$ for all k. The value of the remaining variables is immaterial. From this initial conditions, the link (local) protocol may try to wake the links and it proceeds operating as defined by the algorithm. The SINK comes into operation in state S1, with $n_{SINK} = 0$ and $F_{SINK}(k) = DOWN$ for all k, and proceeds according to the algorithm for the SINK.

3.3: Formal Description of the Algorithms

3.3.1: Introduction

We are now ready to display the formal algorithms performed by each node i in the network. As for the basic algorithms, we present here the algorithms for the two models (message switching and line switching) simultanously and indicate the differences when applicable. The presentation here follows the same lines as the basic algorithms. In addition, in Section 3.3.2 we provide the exact requirements from the local (link) protocol. The "Facts" given in the algorithms are displayed for helping in their understanding and are proven in Theorem 3.2 of Section 3.4.3. A Fact holds if the transition under which it appears is performed.

Definitions of variables

The values a variable can take appear in parentheses.

i = node under consideration.

ℓ = the ℓ-th neighbor of node i: (1,2,...,N];

η = a parameter: (see Theorem 2.2];

n_i = current counter number of node i:(0,1,2,...);

d_i = estimated marginal delay of node i from SINK: $(1,2,\ldots,\infty)$;

b_i = blocking status of node i:(0,1); 0 means not blocked; 1 means blocked;

The processor at node i may receive the following types of messages related to each link (i,ℓ):

MSG(m,d,b,ℓ) = updating message received by i from ℓ:$(m=n_\ell, d=d_\ell, b=b_\ell]$;

FAIL(ℓ) = failure detected on link (i,ℓ);

WAKE(ℓ) = link (i,ℓ) becomes operational, i.e. messages can be sent through it;

We now continue the list of variables:

$F_i(\ell)$ = status of link (i,ℓ) as seen from node i:(UP, DOWN, READY); UP means the link is operational; DOWN means the link is unoperational; READY means the link is ready to be brought up;

$N_i(\ell)$ = the number m received from neighbor ℓ during the current iteration:(NIL, 0,1,...);

$D'_{i\ell}$ = estimated(or calculated) marginal delay on link (i,ℓ): (1,2,...);

$d_i(\ell)$ = last number d received at i from neighbor ℓ:(0,1,2,...,∞);

$D_i(\ell) = d_i(\ell) + D'_{i\ell}$: (1,2,...,$\infty$);

$B_i(\ell)$ = blocking status of neighbor ℓ as known at i:(0,1); 0 means not blocked; 1 means blocked;

$R_i(\ell)$ = status of neighbor ℓ:(NIL, SON); SON means node ℓ is a son of i;

$Z_i(\ell)$ = a synchronization number indicating the iteration number upon which the link (i,ℓ) can be brought up, i.e. changed from READY status to UP status:(0,1,2,...);

mx_i = the largest number m received by node i up to the current time from all neighbors:(0,1,2,...);

CT = a flag indicating the number of transitions the Finite-State-Machine has already performed triggered by the current message: (0,1); 0 means zero transitions; 1 means one or more transitions.

In the formal description that follows, we will need to refer from time to time to certain sets of neighbors. To save space, we define those sets here:

C_i = set of all nodes k s.t. $R_i(k)$ = SON and all other nodes k with $F_i(k)$ = UP, $N_i(k) = mx_i$ and $B_i(k) = 0$.

A_i = set of all nodes k s.t. $R_i(k)$ = SON and all other nodes with $F_i(k)$ = UP and $B_i(k) = 0$.

3.3.2: Properties Required from the Local Protocol

On each link of the network there is a link (local) protocol that is in charge of exchanging messages between neighbors. Our main algorithm assumes that the following properties hold for the local protocol:

1. All links are bidirectional (duplex).
2. $D'_{ik} > 0$ for all links (i,k) at all times. (See (2.9d))
3. If a message is sent by node i to a neighbor ℓ, then in finite time, either the message will be received correctly at ℓ or $F_i(\ell) = F_\ell(i) =$ DOWN. Observe that this assumption does not preclude transmission errors that are recovered by the local protocol (e.g. "resend and acknowledgement").
4. Failure of a node is considered as failure of all links connected to it.
5. A node i comes up in state S3, with $n_i = 0$, $R_i(k) =$ NIL and $F_i(k) =$ DOWN for all links(i,k).
6. The processor at node i receives messages from link (i,ℓ) on a first-in-first-served (FIFO) basis.
7. A link (i,ℓ) is said to have <u>become operational</u> as soon as the local protocol discovers that the link can be used. Links (i,ℓ) and (ℓ,i) become operational at the same time and subject to the following

restrictions, a WAKE "message" is delivered in this case to each of the processors i and ℓ.

WAKE(ℓ) can be received at node i only if

(a) node ℓ receives WAKE(i) at the same (virtual) time;

(b) there are no other outstanding messages on link (i,ℓ) and on (ℓ,i);

(c) $F_i(\ell) = F_\ell(i) = \text{DOWN}$.

8. If $F_i(\ell) = \text{DOWN}$, the only message that the processor at i can receive from ℓ is WAKE(ℓ).

9. (a) If $F_i(\ell) \neq \text{DOWN}$ and $F_\ell(i) \neq \text{DOWN}$ and $F_i(\ell)$ goes to DOWN, then $F_\ell(i)$ goes to DOWN in finite time.

(b) If $F_i(\ell) = F_\ell(i) = \text{DOWN}$ and $F_i(\ell)$ goes to READY, then in finite time, either $F_\ell(i)$ goes to READY or $F_i(\ell) = F_\ell(i) = \text{DOWN}$.

10. When two nodes i and ℓ receive WAKE as described in 7, a "synchronization" between i and ℓ is attempted. At either end the synchronization may or may not be successful (the latter because of a new failure). If it is successful, the node proceeds as in Step A.2 of the formal description. If not, then no action is taken.

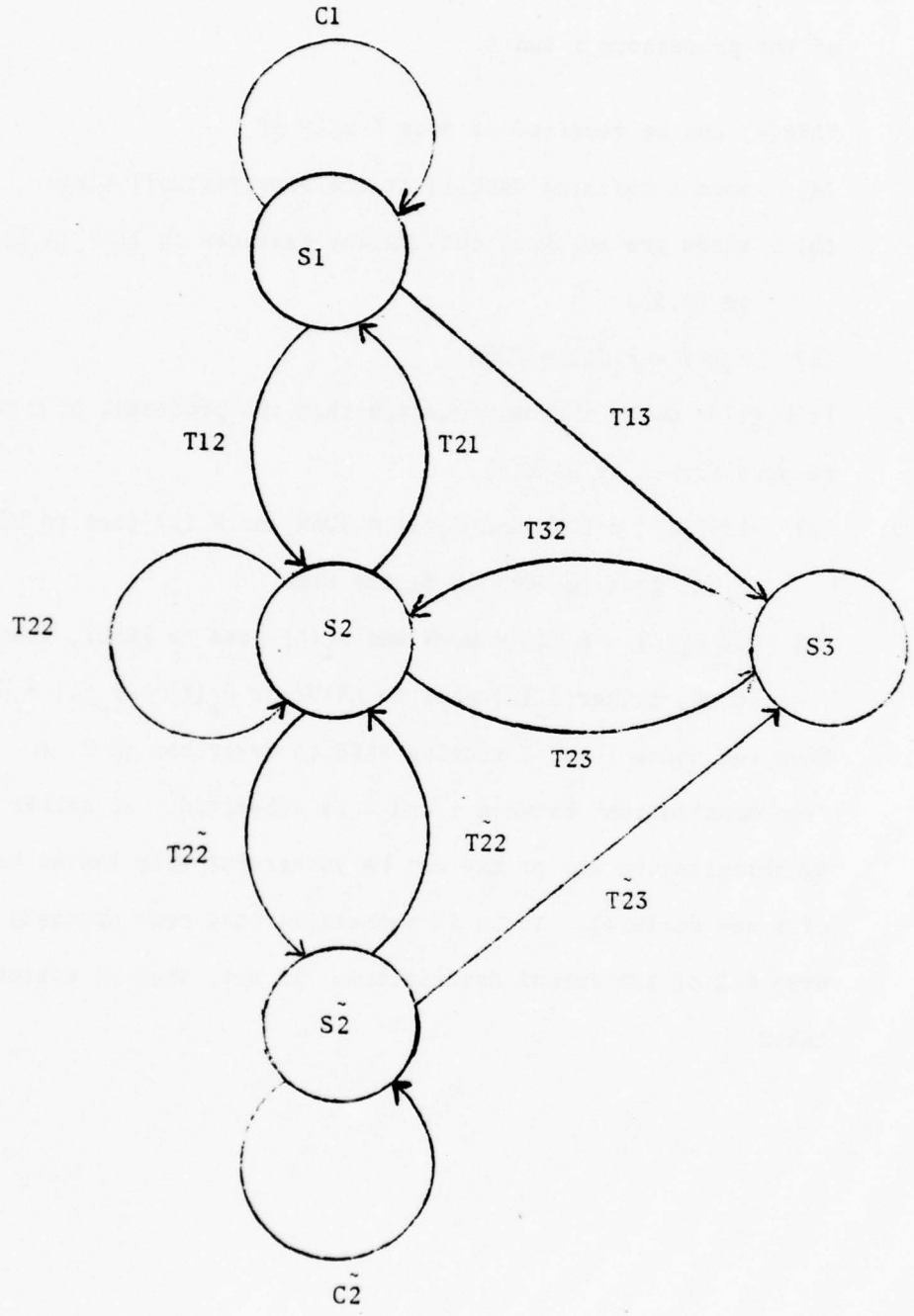


Fig. 5: Finite-State-Machine for an arbitrary node (Extended algorithms).

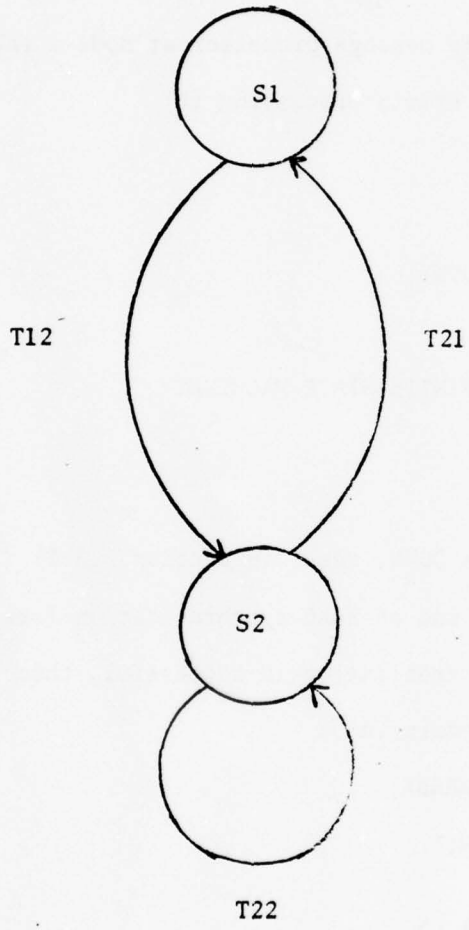


Fig. 6: Finite-State-Machine for the SINK (Extended algorithms).

3.3.3.: Formal Algorithms

(For each node i except the SINK).

A. Operations Done by the Message Processor when a Message is Received

(i.e., when the message processor at node i takes the message from the queue and starts processing it).

A.1 For FALL(ℓ)

A.1.1 $F_i(\ell) \leftarrow$ DOWN;

A.1.2 CT $\leftarrow$ 0;

A.1.3 EXECUTE FINITE-STATE-MACHINE.

A.2 For WAKE(ℓ)

(Fact: $F_i(\ell)$ = DOWN, see 7 in Section 3.3.2)

A.2.1 wait for end of WAKE synchronization, (see 10 in Section 3.3.2) if WAKE synchronization is successful, then

A.2.2 $Z_i(\ell) \leftarrow \max\{n_i, n_\ell\}$;

A.2.3 $F_i(\ell) \leftarrow$ READY;

A.2.4 $N_i(\ell) \leftarrow$ NIL.

A.3 For MSG (m,d,b,ℓ)

A.3.1 if $F_i(\ell)$ = READY, then $F_i(\ell) \leftarrow$ UP (Fact: $m > Z_i(\ell)$);

A.3.2 $N_i(\ell) \leftarrow m$;

A.3.3 $d_i(\ell) \leftarrow d$;

A.3.4 $D_i(\ell) \leftarrow d + D'_{i\ell}$;

A.3.5 $B_i(\ell) \leftarrow b$;

A.3.6 $mx_i \leftarrow \max(m, mx_i)$;

A.3.7 CT ← 0;

A.3.8 EXECUTE FINITE-STATE-MACHINE

B. FINITE-STATE-MACHINE

STATE S1

B.1.1 T12 Condition 12: $\forall k$ s.t. $R_i(k)$ = SON, then $N_i(k)=mx_i$, $D_i(k) \neq \infty$ and $F_i(k)$ = UP;

B.1.2 CT = 0

B.1.3 Fact 12 If MSG, then $m \geq n_i$.

B.1.4 Action 12 For line switched network set

$$d_i \leftarrow \min_{k:k\epsilon C_i} \{D_i(k)\}: \quad (3.1)$$

For message switched network set

$$d_i \leftarrow \sum_{k:R_i(k)=SON} \phi_{ik} D_i(k); \quad (3.2)$$

B.1.5 Check of status: If for any node k s.t. $R_i(k)$=SON then $\{B_i(k) = 1\}$ or

{For line switched network

$$d_i(k) \geq d_i \ \& \ \eta[D_i(k)-d_i] < f_{ik}; \quad (3.3)$$

For message switched network

$$d_i(k) \geq d_i \ \& \ \eta[D_i(k)-d_i]/t_i < \phi_{ik};\} \quad (3.4)$$

then set $b_i \leftarrow 1$; otherwise set $b_i \leftarrow 0$;

B.1.6 $n_i \leftarrow mx_i$;

B.1.7 $\forall k$ s.t. $F_i(k) = \text{READY}$ and $n_i > Z_i(k)$, set $F_i(k) \leftarrow \text{UP}$ and $N_i(k) \leftarrow \text{NIL}$;

B.1.8 $\forall k$ s.t. $F_i(k) = \text{UP}$ and $R_i(k) \neq \text{SON}$, send (n_i, d_i, b_i, i);

B.1.9 $CT \leftarrow 1$

B.2.1 T13 Condition 13: $R_i(\ell) = \text{SON}$;

B.2.2 $\forall k \neq \ell$ s.t. $F_i(k) = \text{UP}$, then $R_i(k) = \text{NIL}$;

B.2.3 MSG(m,d = ∞,b,ℓ) or FAIL(ℓ);

B.2.4 $CT = 0$

B.2.5 Fact 13: If MSG, then $m \geq n_i$.

B.2.6 Action 13: $d_i \leftarrow \infty$;

B.2.7 If MSG, then $n_i \leftarrow m$;

B.2.8 $\forall k$ s.t. $F_i(k) = \text{READY}$ and $n_i > Z_i(k)$, set $F_i(k) \leftarrow \text{UP}$ and $N_i(k) \leftarrow \text{NIL}$;

B.2.9 $\forall k$ s.t. $F_i(k) = \text{UP}$ and $R_i(k) \neq \text{SON}$, send (n_i, d_i, b_i, i);

B.2.10 $R_i(\ell) \leftarrow \text{NIL}$;

B.2.11 Cancel the flow to node ℓ and modify the routing variables by setting $\phi_{i\ell} = 0$ (for message switching) $f_{i\ell} = 0$ (for line switching);

B.2.12 $CT = 1$.

B.3.1 C1 Condition 1: $R_i(\ell)$ = SON;

B.3.2 $\exists\, k \neq \ell$ s.t. $R_i(k)$ = SON and $F_i(k)$ = UP;

B.3.3. MSG (m,d = ∞,b,ℓ) or FAIL (ℓ);

B.3.4 CT = 0

B.3.5 Action 1: $R_i(\ell) \leftarrow$ NIL;

B.3.6 reroute the flow to node ℓ while arbitrarily redistributing it through the remaining sons and modify the routing variables correspondingly.

STATE S2

B.4.1 T21 Condition 21: $\forall k$ s.t. $F_i(k)$ = UP, then $N_i(k) = n_i = mx_i$.

B.4.2 $\exists\, k \in A_i$ s.t. $D_i(k) \leq d_i$;

B.4.3 If CT = 0, then MSG;

B.4.4 $\forall k$ s.t. $R_i(k)$ = SON, then $D_i(k) \neq \infty$.

B.4.5 Fact 21: $d_i \neq \infty$

B.4.6 Action 21: Rerouting;

Caclculate $\alpha = \min_{k:k\in A_i} \{D_i(k)\}$; (3.4)

B.4.7 let k_o be any neighbor that achieves the minimum in (3.4).

B.4.8 For line switched network:

B.4.8.1 If there is any node q s.t. $F_i(q)$=UP with $f_{iq} > 0$, then for all neighbors $k \in A_i$ do:

B.4.8.2 $a_{ik} = D_i(k) - \alpha$;

B.4.8.3 cancel all outgoing links corresponding to incoming links that have been cancelled by fathers; let f'_{ik} be the remaining

B.4.8.4 $\Delta_{ik} = \min\{f'_{ik}, na_{ik}\}$;

B.4.8.5 SET NEW FLOW ($\forall k$ s.t. $F_i(k) = UP$)

$$f_{ik}^{new} = \begin{cases} 0 & k \notin A_i \\ f'_{ik} - \Delta_{ik} & k \in A_i, k \neq k_o \\ f'_{ik} + \sum_{\substack{k \in A_i \\ k \neq k_o}} \Delta_{ik} + \text{any new flow} & k = k_o \end{cases}$$

B.4.8.6 if $f_{ik} = 0$ $\forall k$ s.t. $F_i(k) = UP$, then any new flow is routed through k_o.

B.4.8.7 For <u>message switched</u> network:

B.4.9.1 If $t_i > 0$, then for all neighbors $k \in A_i$ do:

B.4.9.2 $a_{ik} = D_i(k) - \alpha$;

B.4.9.3 $\Delta_{ik} = \min\{\phi_{ik}, na_{ik}/t_i\}$;

B.4.9.4 SET NEW FLOW ($\forall k$ s.t. $F_i(k) = UP$).

$$\phi_{ik}^{new} = \begin{cases} 0 & k \notin A_i \\ \phi_{ik} - \Delta_{ik} & k \in A_i, k \neq k_o \\ \phi_{ik} + \sum_{\substack{k \in A_i \\ k \neq k_o}} \Delta_{ik} & k = k_o \end{cases};$$

B.4.9.5 If $t_i = 0$, then set $\phi_{ik_o} = 1$ and $\forall k \neq k_o$ s.t. $F_i(k) = UP$, set $\phi_{ik} = 0$;

B.4.10 $\forall k$ s.t. $R_i(k) = SON$ send (n_i, d_i, b_i, i);

B.4.11 $\forall k$ s.t. $F_i(k) = UP$, set $R_i(k) \leftarrow NIL$;

set $R_i(k_o) \leftarrow SON$;

$\forall k$ s.t. $k \neq k_o$ and $f_{ik}^{new} > 0$ (for <u>line switched</u> network), $\phi_{ik}^{new} > 0$ (for <u>message switched</u> network), set $R_i(k) \leftarrow SON$;

B.4.12 $\nexists k$ s.t. $F_i(k) = UP$, set $N_i(k) \leftarrow NIL$;

B.4.13 $CT \leftarrow 1$

B.5.1 T22 Condition 22: $\forall k$ s.t. $R_i(k) = SON$, then $N_i(k) = mx_i > n_i$, $D_i(k) \neq \infty$ and $F_i(k) = UP$;

B.5.2 $CT = 0$

B.5.3 Action 22: Same as Action 12.

B.6.1 T22~ Condition 22~: Either same as Condition 1 or Fail (ℓ) s.t. $R_i(\ell) \neq SON$;

B.6.2 $CT = 0$

B.6.3 Action 22~: Same as Action 1 and in addition set $CT \leftarrow 1$.

B.7.1 T23 Condition 23: Same as Condition 13.

B.7.2 Fact 23: Same as Fact 13.

B.7.3 Action 23: Same as Action 13.

STATE S3

B.8.1 T32 Condition 32: $\exists k$ s.t. $F_i(k) = UP$, $mx_i = N_i(k) > n_i$, $D_i(k) \neq \infty$.

B.8.2 Fact 32: $d_i = \infty$, $\nexists k$ s.t. $F_i(k) = UP$, then $R_i(k) = NIL$.

B.8.3 Action 32: Let k^* be a node that achieves

$$\min_{\substack{k:F_i(k)=UP \\ N_i(k)=mx_i}} \{D_i(k)\};$$

B.8.4 If $B_i(k^*) = 1$, then $b_i \leftarrow 1$;

B.8.5 $R_i(k^*) \leftarrow SON$;

B.8.6 $n_i \leftarrow mx_i$;

B.8.7 $d_i \leftarrow D_i(k^*)$;

B.8.8 $\forall k$ s.t. $F_i(k)$ = READY and $n_i > Z_i(1)$
set $F_i(k) \leftarrow$ UP and $N_i(k) \leftarrow$ NIL.

B.8.9 $\forall k$ s.t. $F_i(k)$ = UP and $R_i(k) \neq$ SON,
send (n_i, d_i, b_i, i);

B.8.10 For line switching: any new flow is routed through k^*.
For message switching: set $\phi_{ik^*} \leftarrow 1$;

B.8.11 CT $\leftarrow$ 1

STATE $\tilde{S}2$

B.9.1 $\tilde{T}22$ Condition $\tilde{2}2$: Same as Step B.5.1

B.9.2 Action $\tilde{2}2$: Same as Action 12.

B.10.1 $\tilde{T}23$ Condition $\tilde{2}3$: Same as Condition 13.

B.10.2 Fact $\tilde{2}3$: Same as Fact 13.

B.10.3 Action $\tilde{2}3$: Same as Action 13.

B.11.1 $\tilde{C}2$ Condition $\tilde{2}$: Same as Condition 1.

B.11.2 Action $\tilde{2}$: Same as Action 1.

C. Operation Done by the Message Processor at the SINK

C.1. For FAIL (ℓ)

C.1.1 $F_i(\ell) \leftarrow$ DOWN ; CT=0 EXECUTE FINITE-STATE-MACHINE for SINK.

C.2 For WAKE(ℓ)

(Fact: $F_{SINK}(\ell)$ = DOWN, see 7 in Section 3.3.2)

C.2.1 wait for end of WAKE synchronization (see 10 in Section 3.3.2);
if WAKE synchronization is successfully completed, then

C.2.2 $F_{SINK}(\ell) \leftarrow$ READY.

C.2.3 CT $\leftarrow$ 0;

C.3 For MSG(m,d,b,ℓ)

C.3.1 $N_{SINK}(\ell) \leftarrow m$;

C.3.2. CT ← 0;

C.3.3 EXECUTE FINITE-STATE-MACHINE for SINK.

D. Finite-State-Machine for SINK

STATE S1

D.1.1 T12 Condition 12: Either{CT=0} and {FAIL or WAKE};

D.1.2 or the SINK decides to start a new iteration.

D.1.3 Action 12: If FAIL or WAKE, then $n_{SINK} \leftarrow n_{SINK} + 1$;

D.1.4 $\forall k$ s.t. $F_{SINK}(k)$ = READY, set $F_{SINK}(k) \leftarrow$ UP and $N_{SINK}(k) \leftarrow$ NIL;

D.1.5 $\forall k$ s.t. $F_{SINK}(k)$ = UP, send $(n_{SINK}, 0, 0, SINK)$;

D.1.6 CT ← 1.

STATE S2

D.2.1 T21 Condition 21: $\forall k$ s.t. $F_{SINK}(k)$ = UP, then $N_{SINK}(k) = n_{SINK}$;

D.2.2 MSG;

D.2.3 Action 21: $\forall k$ s.t. $F_{SINK}(k)$ = UP, set $N_{SINK}(k) \leftarrow$ NIL;

D.2.4 CT ← 1;

D.3.1 T22 Condition 22: {CT=0} and {FAIL or WAKE}.

D.3.2 Action 22: Same as Action 12.

3.4: Properties and Validation of the Protocols

3.4.1: Introduction

Some of the properties of the protocols have already been indicated in previous sections. We now turn to state those properties explicitly, along with some others that have not yet been shown.

We begin with some definitions and notations, then we state properties that hold throughout the operation of the network, some of them referring to the entire netowrk at a given instant of time and some to a given node or link as time progresses. Then a series of theorems is stated which enables us to prove the recovery of the network after topological changes. Finally, we show that the extended protocols reduce in fact to the basic protocols in absence of topological changes.

3.4.2: Notations and Definitions

In this subsection, we present several notations and definitions that are used throughout this work. The notations $F_i(k)$, $R_i(k)$, FAIL(ℓ), MSG(m,d,b), n_i, d_i, $N_i(k)$, $D_i(k)$, $Z_i(k)$, S1, S2, $\tilde{S2}$, S3, C1, $\tilde{C2}$, PC(m) and others have been already introduced. We add the time in parentheses whenever we want to refer to the above quantities at a given time t; for instance, $R_i(k)(t)$, $n_i(t)$, $N_i(k)(t)$, etc. We also use the following notations:

SX[n] = state SX with node counter number n.

$s_i(t)$ = state and possibly node counter number n_i of node i at time t. Therefore we sometimes write $s_i(t)$ = S3 for instance and sometimes $s_i(t)$ = S3[n].

SON_i = set of nodes $\{k:R_i(k) = SON\}$. We use either SON_i or $\{k: R_i(k) = SON\}$ at our convenience.

Tψ2[t,i,(n1,n2] means transition to state S2 (from an arbitrary state) occurs at time t at node i; in this transition node i changes its node counter number from n1 to n2. If n1 is arbitrary we write ψ instead of n1.

T21[t,SINK,(n1,n1)] means proper completion of an iteration with counter number n1.

Routing Graph

At a given instant t, a Routing Graph RG(t) is defined as the directed graph whose nodes are the network nodes and whose arcs are given by the pointers $R_i(\ell) = SON$, namely, there is an arc from node i to node ℓ in the Routing Graph RG(t) if and only if $R_i(\ell)(t) = SON$. (in other words $\ell \in SON_i(t)$], or in words if and only if ℓ is a son of i at time t. The graph RG(t) has some very important properties and for describing them, a definition of an order for the states is needed. Therefore, we define that S3 > S2 = $\tilde{S2}$ > S1, and from now on, we agree that Sx $\geq$ Sy means that Sx > Sy or Sx = Sy. We also define the terminating nodes of the RG(t) to be those nodes in the network which have no sons at time t. For instance, Fig. 7b is the Routing Graph of the network in Fig. 7a. Notice that the SINK is always a terminating node.

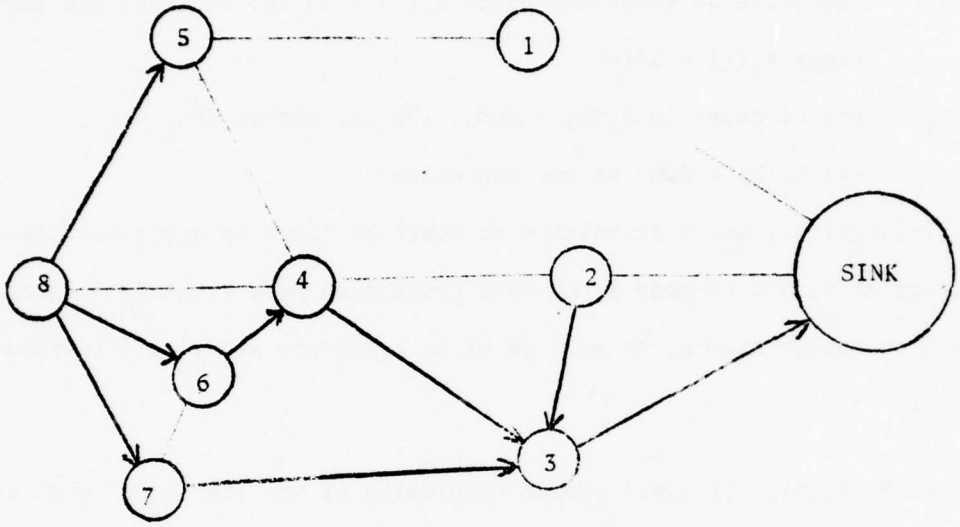


Fig. 7a: Network example (arrows denote sons).

terminating nodes

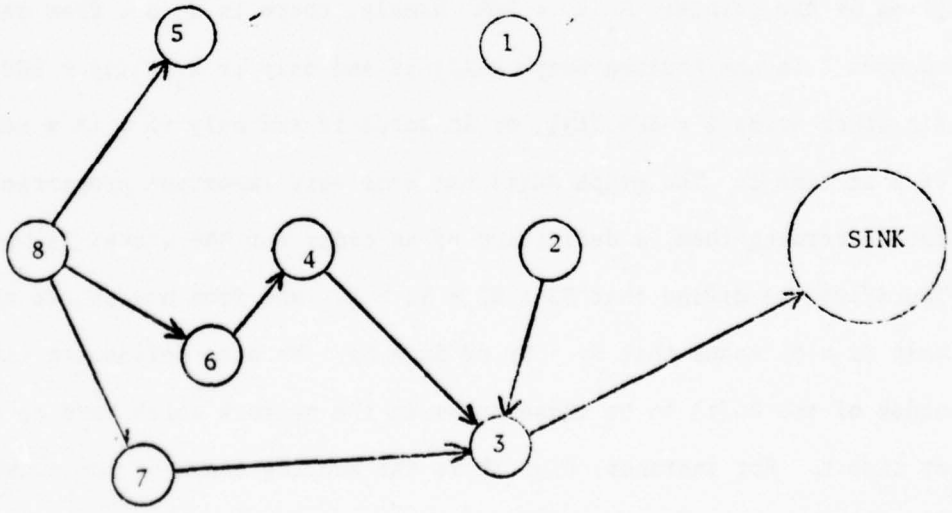


Fig. 7b: Routing Graph for the network in Fig. 7a.

For conceptual purposes, we regard all the actions associated with a transition or a change of the Finite-State-Machine to take place at the time of the transition.

3.4.3: Theorems

Theorem 3.1: (loop-freedom).

At any instant of time t, the RG(t) consists of a loop-free directed pattern (termed lattice from now on) with the following ordering properties:

i) the terminating nodes of the lattice are the SINK and all nodes in S3.

ii) if $\ell \varepsilon SON_i(t)$, then $n_\ell(t) \geq n_i(t)$.

iii) if $\ell \varepsilon SON_i(t)$ and $n_\ell(t) = n_i(t)$, then $s_\ell(t) \geq s_i(t)$.

The proof of Theorem 3.1 is given in Appendix A. According to the theorem, the Routing Graph has at any instant of time the desirable loop-freedom property. It should be noticed here that isolated nodes also belong to the Routing Graph. From the theorem we can realize that certain ordering in the Routing Graph is maintained by the protocols at each instant of time throughout the operation of the network. The order is formed by concatenation of (n_i, s_i) which is nondecreasing when moving from the peripheries towards the terminating nodes of the pattern.

Until now, properties of the entire network at each instant of time throughout the operation of the network, have been stated. In the next theorem we refer to local properties as time progresses.

Theorem 3.2

i) For a given node i, the node counter number n_i is nondecreasing with time and the messages MSG(m,d,b) received from a given neighbor have nondecreasing numbers m.

ii) Between two successive proper completions PC($\bar{m}$) and PC($\bar{\bar{m}}$), for each given m with $\bar{m} \leq m \leq \bar{\bar{m}}$, each node sends to each of its neighbors at most one message MSG(m,d,b) with $d \neq \infty$.

iii) Between two successive proper completions PC($\bar{m}$) and PC($\bar{\bar{m}}$), for each given m with $\bar{m} \leq m \leq \bar{\bar{m}}$, a node enters each of the sets of states {S1[m]},{S2[m],S2[m]},{S3[m]} at most once.

iv) All "Facts" in the formal description of the algorithms in Section 3.3.3 are correct.

A third theorem describes the situation in the network at the time proper completion occurs:

Theorem 3.3

At PC($\bar{m}$), the following hold for each node i:

i) If $n_i = \bar{m}$, then s_i = S1 or s_i = S3.

ii) If a message MSG($\bar{m}$,d,b) with $d \neq \infty$ is on its way to i, then s_i = S3 and $n_i = \bar{m}$.

iii) If either ($n_i = \bar{m}$ and s_i = S1) or $n_i < \bar{m}$, then for all k s.t. $F_i(k)$ = UP it cannot happen that $\{N_i(k) = \bar{m}, D_i(k) \neq \infty\}$.

A combined proof is necessary to show that the properties appearing in Theorems 3.1, 3.2, 3.3 hold. The proof uses a two-level induction, first assuming properties at each proper completion until PC($\bar{m}$) say, hold,

then showing that the other properties hold until the next proper completion named PC($\bar{\bar{m}}$) and finally proving that the necessary properties hold at PC($\bar{\bar{m}}$). The second induction level proves the properties between successive proper completions by assuming that the property holds until just before the current time t and then showing that any possible event at time t preserves the property. The entire rigorous procedure appear in Appendix A.

In order to introduce properties of the protocols regarding normal activity and recovery of the network, the following definitions are necessary:

Definition

We say that a link (i,ℓ) is potentially working if $F_i(\ell) \neq$ DOWN and $F_\ell(i) \neq$ DOWN, and a link (i,ℓ) is working if $F_i(\ell) = F_\ell(i) =$ UP.

Two nodes in the network are said to be potentially connected at time t if there is a sequence of links that are potentially working at time t connecting the two nodes. A set of nodes is said to be strongly connected to the SINK if all nodes in the set are potentially connected to the SINK and for all links (i,ℓ) connecting those nodes, we have either $F_i(\ell) = F_\ell(i) =$ UP or $F_i(\ell) = F_\ell(i) =$ DOWN.

Definition

Consider a given time t, and let ml be the highest counter number of iterations started before t. We say that a pertinent topological change happens at time t if the algorithm at a node i with $n_i(t^-) =$ ml

receives at time t a message FAIL(ℓ) or if WAKE(ℓ) is received at i at time t and the WAKE synchronization is successful. Observe that a pertinent topological change happens if and only if node i has a link (i,ℓ) such that at time t, $F_i(\ell)$ changes from DOWN to READY or from either UP or READY to DOWN.

Theorem 3.4

Let

$L(t)$ = {nodes potentially connected to SINK at time t};

$H(t)$ = {nodes strongly connected to SINK at time t}.

Suppose

$$T\psi 2[t1, SINK, (m1,m1)] \tag{3.12}$$

namely an iteration is started at time t1 with a number that was previously used. Suppose also that no pertinent topological changes have happened while $n_{SINK} = m1$ before t1 and no such changes happen ater t1 for long enough time. Then there exist times t0, t2, t3 with $t0 < t1 < t2 < t3 < \infty$ such that a), b), c), d) hold:

(a) $$T21[t0, SINK, (m1,m1)]; \tag{3.13}$$

(b) $\forall t \in [t0,t3]$, we have $H(t) = L(t) = L(t0)$;

(c) for all $i \in L(t0)$, we have

$$T\psi 2[t2_i, i, (m1,m1)]; \tag{3.14}$$

for some time $t2_i \in [t1,t2]$;

(3.15)

d) i) T21[t3,SINK,(m1,m1)];

ii) RG(t3) for all nodes in L(t0) is a lattice with a single terminating node - the SINK.

In words, Theorem 3.4 dictates that under the given conditions, if a new iteration is started with a number that was previously used, then proper completion with the same number has previously occured and the new iteration will be properly completed in finite time while connecting all nodes of interest (namely, those in L(t0)) to the SINK, both strongly and routingwise. The proof of Theorem 3.4 appears in Appendix B.

The recovery properties of the protocols are described in Theorems 3.5 and 3.6. The proof of Theorem 3.5 appears in Appendix B.

Theorem 3.5

Let L(t), H(t) be as in Theorem 3.4. Suppose

$$T\psi 2[t1, SINK, (m1,m2)] \; ; \; m2 > m1 \; , \tag{3.16}$$

namely an iteration is started at time t1 with a number that was not previously used. Suppose also that no pertinent topological changes happen for a long enough period after t1. Then

a) There exists a time t2, with $t1 \leq t2 < \infty$, such that

i) for all $i \in L(t2)$

$$T\psi 2[t2_i, i, (\psi,m2)]; \tag{3.17}$$

happen at some time $t2_i$ with $t1 \leq t2_i \leq t2$;

ii) $H(t2) = L(t2)$

b) There exists a time $t3 < \infty$ such that

i) $T21[t3, SINK, (m2,m2)]$; (3.18)

ii) $\forall t \in [t2,t3]$, we have $H(t) = L(t) = H(t2)$;

iii) $RG(t3)$ for all nodes in $L(t3)$ is a lattice with a single terminating node - the SINK.

Part (a) of Theorem 3.5 dictates that under the stated conditions, all nodes in L(t2)will eventually enter state S2[m2]. Part b) dictates that the iteration will be properly completed and each node potentially connected to the SINK at time PC(m2) will also have at least one routing path to the SINK.

Finally, we observe that reattachment of a node loosing its only path to the SINK,or leaving state $\tilde{S2}$,or bringing a link up requires an iteration with a counter number higher than the one the node currently has. In the next chapter we present a special protocol that causes such an iteration to be started in finite time. Here only the following assumption is needed:

Assumption A

Suppose that a node i with $n_i(t-) = m$ detects at time t a failure of one of its neighboring links or succeeds in WAKE synchronization with its neighbor ℓ while $Z_i(\ell) = m$ at time t, then the SINK either has started a new iteration with counter number strictly higher than m before t, or will start such an iteration in finite time after t.

Theorem 3.5 and the assumption are combined in the following theorem:

Theorem 3.6 (Recovery)

Let L(t), H(t) be as in Theorem 3.4. Suppose there is a time t1 after which no pertinent topological changes happen in the network for long enough time. Then there exists a time t3 with $t1 \leq t3 < \infty$ such that proper completion happens at t3 and such that all nodes in L(t3) are on a lattice with a single terminating node - the SINK, and are strongly connected to the SINK.

Proof

Let $t0 \leq t1$ be the time of detection of the last pertinent topological change before or at t1. Let node i be the node detecting it and let $m = n_i(t0-)$. Then by assumption, the SINK starts a new iteration with counter number strictly higher than m in finite time. Let $t2 < \infty$ be the time the SINK starts such an iteration with number $m1 > m$. Since by the definition of pertinent topological change, m is the largest number at time t0, we have that $t0 < t2$. By the conditions of this theorem, no pertinent topological changes happen after time t0 for a long enough period, so that no such changes happen after time t2. Consequently Theorem 3.5 holds after this time and the assertion of the Theorem follows.

Q.E.D.

This completes the proof that our extended protocols possess the required properties of being distributed, loop-free and recoverable.

A final theorem is needed for showing that they reduce to the basic protocols after all topological changes have occured

Theorem 3.7 (Optimality)

Let L(t) be as in Theorem 3.4. Suppose there is a time t1 after which pertinent topological changes never happen in the network. Let also the inputs to the network be stationary and let t3 as in Theorem 3.6. Then the network L(t3) will be brought to the minimum average delay over all routing assignments.

Proof

By Theorem 3.6 there exists a time t3 with $t1 \leq t3 < \infty$ such that proper completion happens at t3 and such that all nodes in L(t3) are on a loop-free lattice terminated only at SINK. After time t3 the conditions of Theorem 2.2 hold and the algorithms proceed exactly as the basic algorithms. Therefore, the network L(t3) will be brought to the minimum average delay over all routing assignments.

q.e.d.

CHAPTER 4

ADDITION OF A PROTOCOL FOR TRIGGERING ITERATIONS

4.1: Introduction

In the previous chapter we have described two distributed routing protocols, which are failsafe, namely the protocols operate smoothly under all circumstances. However, to show their ability to cope with topological changes, an assumption has been made, that each time the SINK has to start a new iteration with any specified number, it indeed starts it and does it in finite time. The specific way of triggering a new iteration was of no importance from our point of view as long as the assumption really held.

There exist several procedures for starting a new update iteration and setting the corresponding n_{SINK} in a way that satisfy the above required behavior of the SINK. A simple procedure is that at given intervals of time, or as a result of the detection of a change in the traffic patterns, the SINK increments n_{SINK} and starts a new update iteration. This procedure may make use of a time-out to trigger a new update iteration if the previous one is not properly completed within certain time. If there is a topological change in the network after proper completion, there is no direct triggering of a new update iteration, and thus recovery can be achieved only whenever the SINK decides to start a new update iteration. In addition, this procedure unnecessarily increments n_{SINK} for every update; hence an unnecessarily large number of bits to represent n_{SINK} is required. These two disadvantages are overcome

by the protocol presented in this chapter. This specific protocol, when combined with the protocols described in Chapter 3, enables us to show that whenever need arises,the SINK starts a new update iteration with a specific counter number, within finite-time.

In the following description we first describe the protocol informally, then we combine it with the previous protocols and formally describe the resulted protocols. Finally, an explicit theorem is given that shows the main new property of the resulted protocols.

4.2: Informal Protocol

4.2.1: Introduction

We have observed that loosing a neighbor or bringing a link up requires an iteration with a counter number higher than the number of the node sensing the change. A procedure is therefore needed for each node that senses a topological change to ask the SINK to start a new update iteration with a specified number. Since all our protocols are distributed it would be better to develop a distributed procedure to achieve the desirable goal. Therefore, the following protocol is distributed in nature. In the following description we first show how to ask the SINK a special request and then how to forward this request through the network until it arrives at the SINK.

4.2.2. Request Messages

Any node discovering a topological change by either detecting a failure or sensing that a link is ready to come up generates (in addition to all other operations described in Chapter 3) a special control message - REQ(n_i). The number n_i contained in the message is the current node counter number of the generating node. Since after a topological change, the node usually needs a new update iteration, with a counter number higher than its current number, this message functions as a request message intended for the SINK. Before proceeding to explain how REQ messages are transmitted through the network, let us first assume that such a message is received by the SINK. In such a case, when the SINK receives REQ(m) it immediately starts a new update iteration with counter number (m+1), provided that such an iteration, or an iteration with a number higher than (m+1) has not been previously started. This procedure assures that if all REQ messages generated within the network arrive at the SINK in finite time, then the assumption made at the end of Chapter 3 indeed holds.

In addition, the SINK is allowed to start a new iteration while increasing the counter number at any time.

We now turn to describe how REQ messages are treated and sent by each node. When a message REQ(m) is generated by a node or arrives at a node, it is put in the regular queue and processed on FIFO basis, as all other control messages. When the nodal message processor takes such a message and starts processing it, it first compares its node counter number with the number contained in the message. If it finds that its number is higher than the number contained in the message, then it discards the request message, since it is clear that the requested iteration or even an iteration

with a higher counter number has already been started. Otherwise, namely if the node counter number is equal to or less than the number contained in the message, the request message should be sent forward towards the SINK. Therefore, the protocol dictates that in such a case the node send this message to a specific neighboring node, called preferred son for reasons to be discussed. As much each node i must keep one more memory location denoted by p_i for storage of the identify of its preferred son. The description of how the preferred son is chosen is deferred to the next section. The protocol also dictates that a node that hasn't a preferred son (because of a failure) discards any REQ message it receives.

4.2.3: Selection of the Preferred Son

REQ messages are transmitted through the network along a succession of preferred sons. To insure their arrival at the SINK, the preferred son must be well chosen. The protocol dictates the following way for selecting the preferred son: Each time a node enters state S1 (clearly this may be done only when T21 is performed), it chooses k_o (see B.4.7 in Section 3.3.3) to be its preferred son. Remember that k_o is the only node through which we permit to increase the flow even from zero. As such, it is "preferred" in some sense. In addition, each time a node enters state S2 from any state and hasn't a preferred son, it chooses arbitrarily a node k s.t. $R_i(k)$ = SON, to be its preferred son. This method of selecting the preferred son guarantees that if a REQ(m) message is generated in the network, and sent as described in Section 4.2.2, an iteration with counter number (m+1) or higher, will always be started within finite time. This property is explicitly stated in Theorem 4.2 in Section 4.4 and proved in Appendix B.

4.3: Formal Description

In this section we repeat the formal description of the two algorithms of Section 3.3.3 while adding the protocol we have just described in the necessary places. To save space, we do not copy Section 3.3.3 here again, but only show where the present protocol must be added.

4.3.1: Notations

Two main additional notations are needed for the following description:

p_i = preferred son;

REQ(m) = request message.

4.3.2: The Algorithms

Same as in Section 3.3.3 with the following additions:

After step A.1.3 add:

A.1.4 If $p_i \neq$ NIL, then send REQ(n_i) to p_i.

After step A.2.4 add:

A.2.5 If $p_i \neq$ NIL, then send REQ(n_i) to p_i.

After A.3.8 add:

A.4 For REQ(m)

A.4.1 If $p_i \neq$ NIL and $n_i \leq m$, send REQ(m) to p_i; otherwise discard the message.

After B.1.9

B.1.10 Choose any node k s.t. $R_i(k)$ = SON and set $p_i \leftarrow k$.

After B.2.12 add:

B.2.13 If $p_i = \ell$, then set $p_i \leftarrow$ NIL.

After B.3.6 add:

B.3.7 If $p_i = \ell$, then set $p_i \leftarrow$ NIL.

After B.4.13 add:

B.4.14 Set $p_i \leftarrow k_o$.

After B.8.11 add:

B.8.11 Set $p_i \leftarrow k^*$

After C.3.3 add:

C.4 For REQ(m)

C.4.1 CT ← 0

C.4.2 EXECUTE FINITE-STATE-MACHINE for SINK.

Change D.1.1 to:

D.1.1 T12 Condition 12: {CT = 0} and {FAIL or WAKE or REQ(m=n_{SINK})};

Change D.3.1 to:

D.3.1 T22 Condition 22: {CT = 0} and {FAIL or WAKE or REQ(m=n_{SINK})};

4.4: Properties and Validation of the Protocols

Clearly, the algorithms that are described in Section 4.3.2 are exactly the same as the algorithms of the previous chapter with the simple addition of REQ: Therefore, the present protocols possess the same properties, and all the theorems that are stated in Chapter 3, remain correct here too. In this section we only state the additional properties the protocols have, due to the specific protocol that has been added.

At first, an additional property of the Routing Graph is stated:

Theorem 4.1

The following ordering property is maintained in RG(t) at any instant of time t:

If $p_i(t) \neq$ NIL and $n_{p_i}(t) = n_i(t)$ and $s_{p_i}(t) = s_i(t) =$ S1, then $d_{p_i}(t) < d_i(t)$.

The proof of Theorem 4.1 appears in Appendix A. According to the theorem, in addition to the ordering properties in the Routing Graph that are stated in Theorem 3.1, it has also the following ordering property. For nodes in state S1 with the same node counter number, which are the nodes that have properly completed the update and reroute in a certain iteration, the estimated marginal delays to the SINK are strictly decreasing along the concatenation of preferred sons.

The next theorem comes to substitute the assumption made at the end of Chapter 3.

Theorem 4.2

Suppose that a message REQ(m1) is generated at some time t at some node in the network. Then the SINK has received before t a message REQ(m1) or will receive such a message in finite time after t.

The proof of Theorem 4.2 appears in Appendix B. Before going any further we want to give here an equivalent definition for a pertinent topological change, in connection with request messages.

Equivalent Definition (for pertinent topological changes)

A pertinent topological change happens at time t if and only if a message REQ(m1) is generated at time t, where m1 is the largest update counter number available at time t in the network.

It is easily seen that this definition for a pertinent topological change is equivalent to the definition given in Chapter 3.

Now, it is also clear that the Recovery Theorem 3.6 and the Optimality Theorem 3.7 hold without making the assumption that was needed there, since Theorem 4.2 actually assures the existence of the necessary conditions.

CHAPTER 5

SIMULATION PROGRAM

Actually, there is no need for a simulation program in our work since we validate the protocols analytically. However, there were several problems in expressing the conditions for executing the transitions from one state to another in the Finite-State Machine. To overcome these problems, a simulation program was run. This program simulates the operations done by individual nodes in the network. The details of the program are provided in Appendix C.

As a result of the simulation, Section 3.3.3 was written and property R7 (see Appendix A) was validated. In this chapter we merely give an example that shows the necessity of the simulation program.

Example

Step B.4.3 of Section 3.3.3 was written at first as follows:

B.3.4 MSG; (5.1)

There is need for this step to condition transition T21, otherwise Condition 21 and Condition $\tilde{22}$ may hold at the same time (see step B.6.1 in Section 3.3.3). However, the simulation has shown that (5.1) leads to deadlock in certain circumstances. Here is an example: Let a node i be in state S2 with two neighbors k1 and k2, i.e. $F_i(k1) = F_i(k2) = UP$, and k1 is its only son, i.e. $R_i(k1) = SON$, $R_i(k2) = NIL$. Suppose that $N_i(k2) = mx_i = m1 > n_i$, $D_i(k2) \neq \infty$ i.e. node i has already received a message from neighbor k2. Suppose that

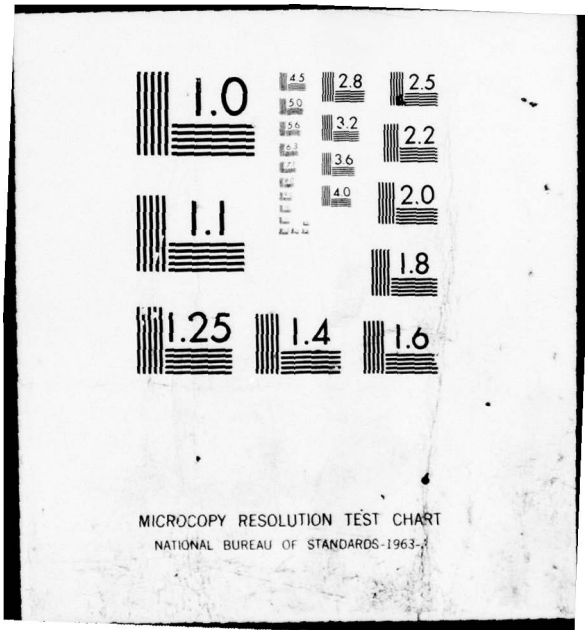
at this point link (i,k1) fails, i.e. a FAIL(k1) is received at i. Then by step B.7.1 of Section 3.3.3 node i goes to S3. By step B.8.1 of Section 3.3.3 it also performs T32 and goes to S2. At this point, no further actions are taken in the Finite-State Machine, particularly, T21 is not performed because of (5.1); however, T21 must be performed at this point, otherwise there is danger that it will never be performed, i.e. deadlock. To overcome this situation, step B.3.4 appears as in Section 3.3.3.

AD-A068 138 MASSACHUSETTS INST OF TECH CAMBRIDGE LAB FOR INFORMA--ETC F/G 17/2
FAILSAFE DISTRIBUTED OPTIMAL ROUTING IN DATA-COMMUNICATION NETW--ETC(U)
MAR 79 M SIDI, A SEGALL N00014-75-C-1183

UNCLASSIFIED LIDS-R-890 NL

2 OF 2
AD
A068138

END
DATE FILMED
6 -79
DDC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

CHAPTER 6

DISCUSSION AND CONCLUSIONS

This thesis presents protocols for constructing and maintaining loop-free routing tables in a data-network, when arbitrary failures and additions happen in the network. In addition, an optimal routing is obtained in steady-state in the sense that the delay is minimized. Several topics involved in these protocols deserve further discussion.

The iteration counter numbers

Evidently, the iteration counter numbers involved in our protocols are increasing infinitely. This does not cause analytic problems, however, it makes difficulties in structured implementation. Therefore other versions of the protocols, in which the iteration counter numbers will be drawn from a finite alphabet, must be considered. Such versions are under current study.

The parameter η

In [GALL 77] and [SEG 77a] it has been proved that the basic protocols converge to the minimum delay in stationary conditions only if the parameter η, involved in the algorithms for each node in the network, is chosen to be very small. Certainly, much larger η's are to be used, in order to allow some dynamics of the routing, so that slowly changing traffic requirements can be followed. In [POU 78] it was shown (by simulation) that large η's still insure convergence.

It is interesting to mention here that if optimality is not seeked and if we allow very large n's ($n \to \infty$), then our extended protocols reduce to the protocol of [SEG 77c], since at each iteration only one son can be chosen.

The request protocol

In Chapter 4 we described a very simple protocol for triggering new iterations when need arises (because of topological changes). Though the protocol is simple, the proofs of its correctness are very complicated. Furthermore, we couldn't validate the protocol unless we assumed that the d's (marginal delay of the nodes) are non-negative integers. Therefore, other protocols which are simple and at the same time can be easily validated must be considered.

State $\tilde{S2}$

State $\tilde{S2}$ of the Finite State-Machine was introduced in order to prevent T21 from happening at a node that is in state S2 and discovers a failure on one of its links or receives MSG($d = \infty$) from a nonsingle son. This avoids nodes to update routes based upon invalid information. In addition, this precludes proper completion from happening, thus enables us to validate the request protocol seperately from all other proofs. However, it is easily noticed that S2 is artificial and actually unnecessary. Step B.4.2 in Section 3.3.3 insures that T21 is prevented, if there is danger that the nodes will update routes based upon invalid information.

We have not omitted this state ($\tilde{S2}$) since we feel it gives better and easier understanding of the operations done by each node, and it does

APPENDIX A

This appendix is organized as follows: We start with several notations that are used in the following proofs, then we proceed with the statements of a few properties that follow immediately from the formal description given in Sections 3.3.3 and 4.3.2. Lemmas A.1 - A.5 and Theorem A.1 contain the proofs of Theorems 3.1, 3.2, 3.3 and 4.1, together with some other properties needed in the proofs themselves. For simplicity, we use in the appendices the word "algorithms" to describe Sections 3.3.3 and 4.3.2.

Notations

In addition to all the notations we have already introduced , we use a compact notation to describe changes accompanying a transition, as follows:

Txy[t,i,MSG(m1,d1,b1,ℓ1), SEND(m2,d2,b2,ℓ2), (n1,n2), (d1,d2), (SON1,SON2), (p1,p2), (mx1, mx2)] (A.1)

will mean that transition from state Sx to state Sy occurs at time t at node i caused by receiving MSG(m1,d1,b1) from neighbor ℓ1; in this transition i sends MSG(m2,d2,b2) to neighbor ℓ2, changes its node counter number n_i from n1 to n2, its estimated marginal delay to the destination d_i from d1 to d2, its set SON_i of sons from SON1 to SON2, its preferred son p_i from p1 to p2 and the largest update counter number received up to now mx_i from mx1 to mx2.

Similarly,

$$Txy[t,i,FAIL(\ell 1),SEND(m2,d2,b2,\ell 2),(n1,n2),(d1,d2),(SON1,SON2),(p1,p2),(mx1,mx2)] \quad (A.2)$$

denotes the same actions as above, except that they are caused by receiving FAIL message from neighbor $\ell 1$.

Another compact notation is used to describe changes which are not accompanied by a transition and are done in the Finite-State-Machine, as follows:

$$Cx[t,i,MSG(m,d,b,\ell),(SON1,SON2),(p1,p2)] \quad (A.3)$$

will mean that a change is caused by receiving a message MSG(m,d,b) from neighbor ℓ ; in this change the set SON_i of sons is changed from SON1 to SON2 and the preferred son p_i is changed from p1 to p2.
Similarly,

$$Cx[t,i,FAIL(\ell),(SON1,SON2),(p1,p2)] \quad (A.4)$$

means the same, except that the change is effected by receiving FAIL message from neighbor ℓ.

For simplicity, all arguments in the above notations that are of no interest in a given description are supressed, and if for example n1 is arbitrary then $(\psi,n2)$ is written instead of (n1,n2). Similarly, if one of the states is arbitrary, ψ will replace this state.
In particular observe that

$T\psi 2[t,SINK,(\psi,n2)]$ (A.5)

means that an iteration with counter number n2 is started at time t and

$T21[t,SINK,(n2,n2)]$ (A.6)

means that proper completion of the iteration occurs at time t.

For Txy[t] or Cx[t] we also use the following notations:

t- = time just before the transition or the change.

t+ = time just after the transition or the change.

Also

$[t,i,MSG[m,d,b,\ell)]$ (A.7)

is used to denote the fact that a message MSG(m,d,b) is received at time t at node i from node ℓ, whether or not the receipt of the message causes a transition or a change.

Similarly

$[t,i,FAIL(\ell)]$ (A.8)

is used.

Properties of the Algorithms

R1 Any change in n_i, s_i, or sending any message MSG(m,d,b) can happen only while node i performs a transition. A change in SON_i can happen only while node i performs a transition Txy or a change Cx.

R2 Txy [t,i,SEND(m,d,b),(ψ,n2),(ψ,d2),(ψ,mx2)] implies d=d2.

If $d \neq \infty$, then

i) Txy = T21 or Tψ2

ii) n2 = m = mx2

If $d = \infty$, then

iii) Txy = Tψ3

iv) n2 = m

R3 T32[t,i,(n1,n2)] $\Rightarrow$ n2 > n1.

T22[t,i,(n1,n2)] $\Rightarrow$ n2 > n1

$\tilde{t}22$[t,i,(n1,n2)] $\Rightarrow$ n2 > n1

R4 $s_i(t) = S3 \Rightarrow \nexists k$ s.t. $R_i(k)(t) = SON \Rightarrow SON_i(t) = NIL \Rightarrow d_i(t) = \infty$

R5 Txy[t,i,(SON1,SON2)], SON1 ≠ NIL, SON1 ≠ SON2 $\Rightarrow$ Txy = Tψ3 or T21 or T22

R6 $mx_i(t)$ is nondecreasing as time increases for any node i.

R7 In the Finite-State-Machine, no two conditions can hold at the same time. This implies that the order of checking the conditions of the transitions and changes is irrelevant.

R8 For all t and all nodes i in the network, $n_{SINK}(t) \geq n_i(t)$ and $n_{SINK}(t) \geq mx_i(t)$.

R9 T32[t,i,(NIL,SON2),(NIL,p2),(∞,d2)] implies:

(i) $p2 \in SON2$; (ii) $d2 > d_{p2}(t)$.

R10 The Finite-State-Machine has two types of transitions. The first type is effected directly by the incoming message, while the second type is caused by the situation in the memory of the node. Each message can trigger only one transition of the first type, and this transition comes always before transitions of the second type. This is controlled by the variable CT in Section 3.3.3. Transitions $\tilde{T22}$, T21 and T32 are of the second type, transitions T13, $\tilde{T23}$, $\tilde{T23}$, $\tilde{T22}$ and the changes C1 and $\tilde{C2}$ are of the first type. Transitions T12 and T22 belong to both types.

R11 The possible changes of $F_i(\ell)$ are given in Fig. 8. The types

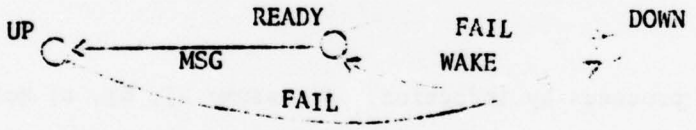


Fig.8: Possible changes of $F_i(\ell)$.

of messages causing them are also shown. A pertinent topological change happens if $F_i(\ell) \to$ DOWN or $F_i(\ell)$ changes from DOWN to READY at a node i with $n_i(t-) = m1$, where m1 is the highest counter number of iterations started before t.

The following lemma proves statement i) of Theorem 3.2 and shows the role of the node counter number n_i. Here we see for the first time that several properties have to be proved in a common induction.

Lemma A.1

a) Let

$$[t1,i,MSG(m1,d1,b1,\ell)], \tag{A.9}$$

$$[t2,i,MSG(m2,d2,b2,\ell)], \tag{A.10}$$

then $t2 > t1$ implies $m2 \geq m1$.

b) For a node i, n_i is nondecreasing with time.

c) Let $M_i(t,\ell)$ denote the counter number m of the last message MSG(m,d,b) received at node i before or at time t from node ℓ. Then

$$n_i(t) \leq M_i(t,\ell) \quad \forall \ell \in SON_i(t) \tag{A.11}$$

Proof

The proof proceeds by induction. We assume a), b), c) hold up to time t- for all nodes in the network. We then prove that any possible event at time t preserves the properties. This combined with the fact that a), b), c) hold trivially at the time any node comes up for the first time completes the proof.

a) Suppose $t = t2$.

Then by FIFO and property R2 it is clear that:

$$\exists\, t3 \text{ s.t. } Txy[t3,\ell,SEND(m1,d1,b1)] \Rightarrow n_\ell(t3) = m1 \tag{A.12}$$

$$\exists\, t4 \text{ s.t. } T\alpha\beta[t4,\ell,SEND(m2,d2,b2)] \Rightarrow n_\ell(t4) = m2 \tag{A.13}$$

with $t3 < t4 < t$.

By induction hypothesis on b) n_ℓ was nondecreasing up to (but not including) time t, so $m1 \leq m2$.

q.e.d.

b) Here we check all possible events at time t.

- If $C\psi[t,i,(n1,n2)]$ or $T21[t,i,(n1,n2)]$ or $\tilde{T22}[t,i,(n1,n2)]$ happens, then the node counter number n_i is not changed so $n1 = n2$ (see Action 1, Action $\tilde{2}$, Action 21, Action $\tilde{22}$ in Section 3.3.3), q.e.d.
- If $T32[t,i,(n1,n2)]$ or $T22[t,i.(n2,n2)]$ or $\tilde{T22}[t,i,(n1,n2)]$ happens, then by property R3, $n2 > n1$, q.e.d.
- If $T12[t,i,(n1,n2),(SON1,SON2)]$ happens, then by induction hypothesis on c)

  $$n1 \leq M_i(t-,k) \quad \forall k \in SON1.$$

  Since in T12 we have SON1 = SON2, then

  $$n1 \leq M_i(t-,k) \quad \forall k \in SON2.$$

  By applying a) at time t we get:

  $$M_i(t-,k) \leq M_i(t+,k) = n2 \quad \forall k \in SON2$$

  where the last equality follows from steps B.1.1, B.1.6 in Section 3.3.3

  Hence $n1 \leq n2$, q.e.d.
- If $T\psi3[t,i,(n1,n2),(SON1,SON2)]$ happens, then the transition might be caused by either FAIL(ℓ) or MSG(m,d,b,ℓ).

  If FAIL(ℓ), then n2 = n1 (see step B.2.7 in Section 3.3.3), q.e.d.

  If MSG(m,d,b,ℓ) then from steps B.2.1, B.2.7 in Section 3.3.3 we know that $\ell \in SON1$ and n2 = m, therefore:

  $$n1 \leq M_i(t-,\ell) \leq m = n2$$

  where the inequalities follow respectively from induction hypothesis on c) and by applying a) at time t, q.e.d.

c) Here we check again all possible events at time t.

- If $C\psi[t,i]$ or $\tilde{T22}[t,i]$ or a received message causes no transition, then from Section 3.3.3 we have:

  $$n_i(t+) = n_i(t-) \text{ and } SON_i(t-) \supseteq SON_i(t+)$$

  From induction hypothesis on c)

  $$n_i(t-) \leq M_i(t-,k), \quad \forall k \in SON_i(t-).$$

  Therefore:

  $$n_i(t+) \leq M_i(t-,k), \quad \forall k \in SON_i(t+).$$

  Finally, by applying a) at time t we get:

  $$n_i(t+) \leq M_i(t-,k) \leq M_i(t+,k), \quad \forall k \in SON_i(t+), \qquad \text{q.e.d.}$$

- If $T\psi3[t,i]$ happens, then $SON_i(t+) = NIL$, so nothing has to be proved.

- If $T21[t,i]$ happens, then the counter number of the last message received before time t from any neighbor is (see step B.4.1 in Section 3.3.3)

  $$n_i(t-) = n_i(t+) = mx_i(t-)$$

  therefore,

  $$n_i(t+) = M_i(t+,k), \quad \forall k \in SON_i(t+], \qquad \text{q.e.d.}$$

- If $T\psi2[t,i]$ happens, then (see B.1.1, B.5.1, B.8.1, B.9.1 in Section 3.3.3)

  $$n_i(t+) = mx_i(t-) = N_i(k)(t-) = M_i(t+,k), \quad \forall k \in SON_i(t+). \quad \text{q.e.d.}$$

The next lemma shows what are the messages that can travel on a link after a failure or after a message with $d = \infty$.

Lemma A.2

a) If

$$[t1,i,MSG(m1,d1,b1,\ell)], \tag{A.14}$$

$$[t2,i,MSG(m2,d2,b2,\ell)] \tag{A.15}$$

where $t2 > t1$, $d1 = \infty$, then $m2 > m1$.

b) If

$$[t1,i,FAIL(\ell)], \tag{A.16}$$

$$[t2,i,MSG(m2,d2,b2,\ell)] \tag{A.17}$$

where $t2 > t1$, then $m2 > n_i(t1)$ and also $m2 > n_\ell(t1)$.

Proof

a) $\exists t3 < t1$ such that

$$T\psi3[t3,\ell,SEND(m1,d1=\infty,b1,i),(\psi,n2)] \tag{A.18}$$

and from property R2 we have $m1 = n2$.

The next transition of node ℓ must be:

$$T32[t4,\ell,(n2,n3)] \tag{A.19}$$

with $n3 > n2$, so that by Lemma A.1 b) which says that n_ℓ is nondecreasing, we see that ℓ will never send any message $MSG(m,d,b)$ with $m \leq m1$ after t3. R2 and FIFO at node i completes the proof, q.e.d.

b) After a failure, a link (i,ℓ) can be brought up only with numbers strictly higher than $Z_i(\ell) = \max(n_i,n_\ell)$. Since n_i and n_ℓ are nondecreasing numbers by Lemma A.1 b), the proof is completed, q.e.d.

Lemma A.3

If

$$T\psi 2[t1,i,(\psi,m)] \tag{A.20}$$

then $\forall t > t1$ we have for all k s.t. $F_i(k)(t) = \text{READY}$ that $Z_i(k)(t) \geq m$. Therefore, no link is brought up by node i with number m after entering S2[m] (brought up means $F_i(k) \leftarrow \text{UP}$).

Proof

If at time t1- we have $F_i(k)(t1-) = \text{READY}$ and $Z_i(k)(t1-) < m$, then link (i,k) is brought up by node i ($F_i(k) \leftarrow \text{UP}$) at time t1(see B.1.7, B.2.8, B.5.4, B.9.8, B.9.2 in Section 3.3.3).

If at time t1- we have $F_i(k)(t1-) = \text{READY}$ and $Z_i(k)(t1-) \geq m$, then nothing would happen at time t1 and for all $t > t1$ $Z_i(k)(t) \geq m$, since $Z_i(k)$ is nondecreasing (by Lemma A.1 b)).

If $F_i(k)$ has been set to READY after t1, then by Lemma A.1 b)

$$n_i(t) \geq m \qquad \forall t > t1$$

and clearly $Z_i(k)(t) \geq m \quad \forall t > t1$, q.e.d.

Lemma A.4

If $F_i(\ell)(t) = \text{READY}$ and

$$[t,i,\text{MSG}(m,d,b,\ell)], \tag{A.21}$$

then $m > Z_i(\ell)(t)$. Observe that this is the Fact in step A.2 in Section 3.3.3.

Proof

From steps A.1, A.2, A.3 in Section 3.3.3 and property 7 in Section 3.3.2, $F_i(\ell)$ can go the READY only from DOWN and only when successful synchronization of WAKE(ℓ) occurs at i. Let t1 < t be the last time it occurs. By property 7 in Section 3.3.2, at time t1 there are no outstanding messages on (i,ℓ) or (ℓ,i) and $Z_i(\ell)$ is established as $\max\{n_i,n_\ell\}$ (see A.2.2 in Section 3.3.3). Therefore, the message in (A.21) must have been sent at time t2 > t1 and since node ℓ sends messages only to nodes k for which $F_i(k)$ = UP, it follows that $F_i(\ell)(t2+)$ = UP. But $F_i(\ell)$ could have been set to UP only from READY because of B.1.7, B.2.8, B.5.3, B.7.3, B.8.8, B.9.2 or B.10.3 in Section 3.3.3 and not because of A.3.1 and in all the above cases we have $n_\ell > Z_\ell(i) = Z_i(\ell)$. Since n_ℓ is nondecreasing and ℓ sends MSG(m,d,b) only with $m = n_\ell$, the assertion follows, q.e.d.

Lemma A.5

All "Facts" in Section 3.3.3 are correct.

Proof

The Fact appearing in step A.2 in Section 3.3.3 is proved in Lemma A.4. The Fact in A.3.1 follows from property 7 in Section 3.3.2. Fact 32 is correct since from B.2.2, B.2.6, B.2.10, B.7.1, B.7.3, B.10.1, B.10.3 in Section 3.3.3 we conclude that

$T\Psi 3[i,(d1,d2),(SON1,SON2)]$ (A.22)

implies d2 = ∞, SON2 = NIL.

Facts 13,12,23 and $\tilde{23}$ follow from Lemmas A.1 a) and A.1 c), since if MSG(m,d,b) is received at node i at time t from node ℓ and Tψ3 or T12 happen, then $\ell \in SON_i(t-)$ and

$m \triangleq$ number received at time t by node i from $\ell \geq M_i(t-,\ell) \geq n_i(t-)$.

Fact 21 is correct, since if

Tψ2[i,(d1,d2)]

happens, then $d2 \neq \infty$ and since $SON_i = NIL$ iff $s_i = S3$, q.e.d.

The next Theorem completes the proofs of Theorems 3.1, 3.2, 3.3 and 4.1.

Theorem A.1

Let PC($\bar{m}$) and PC($\bar{\bar{m}}$) be the instants of occurence of two successive proper completions. Then,

a) Theorem 3.3.

b) Consider any number $m1 \leq \bar{m}$. Let $\tilde{m}$ be the highest counter number $\tilde{m} \leq m1$ such that PC($\tilde{m}$) occurs. Let LPC($\tilde{m}$,m1) be the time of occurrence of the last PC($\tilde{m}$) such that PC($\tilde{m}$) $\leq$ PC($\bar{m}$). If for any i,k, $t \leq$ PC($\bar{\bar{m}}$), we have either

$$N_i(k)(t) = m1 = \tilde{m},\ D_i(k)(t) \neq \infty,\ s_i(t) \neq S3,\ n_i(t) = \tilde{m} \qquad (A.23)$$

or

$$N_i(k)(t) = m1 > \tilde{m} \qquad (A.24)$$

then $\exists \tau1 \in [LPC(\tilde{m},m1),t)$ and $\exists \tau2 \in (\tau1,t)$ such that

$$[\tau1,k,SEND(m1,d1,b1,i)] \qquad (A.25)$$

$$[\tau2,i,MSG(m1,d1,b1,k)] \qquad (A.26)$$

with $d1 = D_i(k)(t) - D'_{ik}(\tau 2)$.

(Note: In words, the above insures that a message MSG(m1,d1,b1) was sent and received no earlier than $LPC(\tilde{m},m1)$].

c) Consider, any number $m1 \leq \bar{\bar{m}}$. Let $\tilde{m}$ be the highest counter number $\tilde{m} \leq m1$ such that $PC(\tilde{m})$ occurs. Let $LPC(\tilde{m},m1)$ be the time of occurrence of the last $PC(\tilde{m})$ such that $PC(\tilde{m}) \leq PC(\bar{\bar{m}})$. Then

i) $[t1,i,MSG(m1,d1,b1,\ell)]$, (A.27)

$[t2,i,MSG(m2,d2,b2,\ell)]$, $d2 \neq \infty$ (A.28)

where $LPC(\tilde{m},m1) \leq t1 < t2 \leq PC(\bar{\bar{m}})$ imply $m2 > m1$

ii) If

$T21[t1,i,(n1,n1)]$, $n1 = m1$ (A.29)

$[t2,i,MSG(m,d,b,\ell)]$, $d \neq \infty$ (A.30)

where $LPC(\tilde{m},m1) \leq t1 < t2 \leq PC(\bar{\bar{m}})$,

then $m > n1$.

iii) A node i enters, between $LPC(\tilde{m},m1)$ and $PC(\bar{\bar{m}})$, each of the following sets of states at most once:

$\{S1[m1]\},\{S2[m1],\tilde{S2}[m1]\},\{S3[m1]\}$.

(Note: Observe that for the particular case where $\tilde{m} = \bar{m}$ this is Theorem 3.2 iii)).

d) i) The possible transitions or changes at a node are the following , where $n2 \geq n1$ and $n3 > n1$: $T12[(n1,n2)]$, $T13[(n1,n2)]$, $C1[(n1,n1)]$, $T21[(n1,n1)]$, $\tilde{T22}[(n1,n1)]$, $T23[(n1,n2)]$, $T22[(n1,n3)]$, $\tilde{T22}[(n1,n3)]$, $\tilde{T23}[(n1,n2)]$, $\tilde{C2}[(n1,n1)]$, $T32[(n1,n3)]$ and $C3[(n1,n1)]$.

ii) $T21[t,i,(n1,n1)]$ implies that $\forall k$ s.t. $i \in SON_k(t)$ then (A.31)

$s_k(t) = S1[n1]$.

e) Theorem 3.1 and Theorem 4.1.

f) i) Suppose

$$T21[t,i,(n1,n1)] \tag{A.32}$$

happens with $n1 = \bar{\bar{m}}$, and let $\tau 1$ be the last time before t such that

$$T\psi 2[\tau 1,i,(\psi, n1)] \tag{A.33}$$

happens. Then we have $F_i(k)(\tau 1) = UP$ if and only if $F_i(k)(\tau) = UP$ $\forall \tau \in [\tau 1,t]$.

ii) If for some $t \in (PC(\bar{m}),PC(\bar{\bar{m}})]$ we have

$$T\psi 2[t,i,(\psi,n2)],\ n2 = \bar{\bar{m}} \tag{A.34}$$

Then

$$\exists \tau 1 \in (t,PC(\bar{\bar{m}})] \text{ such that } T21[\tau 1,i,(n2,n2)] \text{ happens,} \tag{A.35}$$

and $\nexists \tau 2 \in [t,PC(\bar{\bar{m}})]$ such that $T23[\tau 2,i]$

or $T22[\tau 2,i]$ happen

g) If $\exists i,k$, $t \in (PC(\bar{m}),PC(\bar{\bar{m}})]$ such that for some $\tau \in (PC(\bar{m}),t)$ holds

$$[\tau,k,SEND(\bar{\bar{m}},d,i)] \ , \ d \neq \infty \tag{A.36}$$

and if node i either has not received this message by time t, or has $N_i(k)(t) = \bar{\bar{m}}$, $D_i(k)(t) \neq \infty$, then $\exists t1 \in [t,PC(\bar{\bar{m}})]$ such that

$s_i(t1) = S2[\bar{\bar{m}}]$ <u>or</u> $s_i(t1) = S3[\bar{\bar{m}}]$.

Proof

a) As said before, the proof proceeds using a two-level induction. We first notice that a) trivially holds at the time the network comes up for the first time (this time might be denoted by PC(0)). Then we assume that a)-g) hold at every time up to and including PC($\bar{m}$). Next we prove that b)-g) hold until the next proper completion PC($\bar{\bar{m}}$), using the second level induction. Finally, we show that a) holds at PC($\bar{\bar{m}}$), thus completing the proof.

b) Clearly, a message MSG(m,d,b) with m = $N_i(k)(t)$, must have been sent before t. We have to show that such a message must have been sent after LPC($\tilde{m}$,ml).

For the case (A.24) where $N_i(k)(t) > \tilde{m}$, suppose the message has been sent before LPC($\tilde{m}$,ml), then it implies by R2 that at LPC($\tilde{m}$,ml) we have $n_k > \tilde{m}$ contradicting R8 and implying such a message has been sent and therefore received after LPC($\tilde{m}$,ml).

For the other case (A.23) where $N_i(k)(t) = \tilde{m}$, $D_i(k)(t) \neq \infty$, $s_i(t) \neq S3$, $n_i(t) = \tilde{m}$, assume that the message MSG(m,d,b) has been sent by k to i before LPC($\tilde{m}$,ml) and no such a message has been sent by k to i afterwards. First assume the message is on its way to node i at LPC($\tilde{m}$,ml). This implies by the induction hypothesis on a) ii) applied at LPC($\tilde{m}$,ml) that we have $s_i(\text{LPC}(\tilde{m},\text{ml})) = S3$ with $n_i = \tilde{m}$. However, at time t, $s_i(t) \neq S3$ and when a node leaves state S3 it strictly increases its node counter number, but we have $n_i(t) = \tilde{m}$, contradicting the assumption. Next assume the message has arrived at node i before LPC($\tilde{m}$,ml). Since the

situation $N_i(k) = \tilde{m}$, $D_i(k) \neq \infty$ holds until time t it also holds at LPC($\tilde{m}$,m1). At LPC($\tilde{m}$,m1) $n_i \leq \tilde{m}$. If $n_i = \tilde{m}$ at LPC($\tilde{m}$,m1) then by the induction hypothesis on a) i) applied at LPC($\tilde{m}$,m1) we have s_i = S1 (we have already seen that s_i(LPC($\tilde{m}$,m1)) $\neq$ S3. In either cases, {$n_i = \tilde{m}$ & s_i = S1} or {$n_i < \tilde{m}$} by the induction hypothesis on a) iii), for all k s.t. $F_i(k)$ = UP it cannot happen that {$N_i(k) = \tilde{m}$, $D_i(k) \neq \infty$} at LPC($\tilde{m}$,m1), asserting a contradiction. Therefore, (A.25) is asserted. q.e.d.

c) Suppose c)i)ii) and iii) are true <u>for all nodes in the network</u> up to time t-. We prove c) i) and c) ii) for t2 =t and then prove c) iii) for t.

i) If d1 = ∞, then m2 > m1 by Lemma A.2 a). From Lemma A.1 a) m2 $\geq$ m1. So, assume d1 $\neq \infty$ and m2 = m1 and we are going to show that this assumption asserts a contrdiction.

If d1 $\neq \infty$ and m2 = m1, then Lemma A.2 a) and Lemma A.1 a) respectively, imply that $\nexists$ t3 $\in$ (t1,t2] such that

[t3,i,MSG(m,d=∞,b,ℓ)] (A.37)

or such that

[t3,i,MSG(m3,d3,b3,ℓ)] (A.38)

with m3 $\neq$ m2 = m1. Therefore the two messages received at t1 and t = t2 can be taken as consecutive. So using b), FIFO and property R1 it turns that $\exists$ t4 $\in$ [LPC($\tilde{m}$,m1),t1] and $\exists$ t5 $\in$ (t4,t2] such that

Txy[t4,ℓ,SEND(m1,d1,b1,i)], d1 $\neq \infty$; (A.39)

T$\alpha\beta$[t5,ℓ,SEND[m1,d2,b2,i)], d2 $\neq \infty$. (A.40)

By R2, Txy = T21 or Tψ2 and same for Tαβ. But by induction hypothesis on c) iii), node ℓ cannot enter {S2[m1],$\tilde{S2}$[m1]} twice between LPC($\tilde{m}$,m1) and PC($\bar{\bar{m}}$), so that the only possibilities are:

{Tψ2[t4,ℓ]} AND {T21[t5,ℓ]}

and no other transition happens between t4 and t5. However, in Tψ2[t4,ℓ], node ℓ sends a message to every neighbor except sons, i.e. except those nodes that belong to $SON_i(t4+)$ (see steps B.1.8, B.5.3, B.8.9, B.9.2 in Section 3.3.3), and in T21[t5,ℓ], only to sons, i.e. to nodes that belong to $SON_i(t5-)$ (see B.4.10 in Section 3.3.3). Since no other transition happens between t4 and t5 we have $SON_i(t4+) = SON_i(t5-)$, contradicting (A.39), (A.40).

So, m2 > m1 q.e.d.

ii) Clearly, $F_i(\ell)(t2-) = UP$. If $F_i(\ell)(t1) \neq UP$, then Lemma A.4 together with the facts that n_i is nondecreasing (by Lemma A.1b) and that $Z_i(\ell)$ is established as in step A.2.2 in Section 3.3.3, show that the first message MSG(m2,d,b,ℓ) that can be received by node i from node ℓ after t1 must have m2 > m1 = n1. Then the assertion follows from Lemma A.1 a).

Suppose now that $F_i(\ell)(t1-) = UP$, then step B.4.1 in Section 3.3.3 requires

$N_i(\ell)(t1-) = n1 = m1$

and by the definition of LPC($\tilde{m}$,m1) we have n1 = m1 ≥ $\tilde{m}$. We now distinguish between two cases:

If $D_i(\ell)(t1-) = \infty$, then ∃t3 < t1 (possibly t3 < LPC($\tilde{m}$,m1)) such that

$$[t3,i,MSG(n1,d1=\infty,b,\ell)] \qquad (A.41)$$

and the assertion follows from Lemma A.2. a).

If $D_i(\ell)(t1-) \neq \infty$, then from b) it follows that

$\exists\, t3 \in [LPC(m,m1),t1)$ such that

$$[t3,i,MSG(n1,d1,b,\ell)] \ , \quad d1 \neq \infty \qquad (A.42)$$

and the assertion then follows from c) i).

iii) From Lemma A.1 b), n_i is nondecreasing, so that once n_i is increased, it cannot be returned to the old value.

From Section 3.3.3, a node can leave $\{S2[m1],\tilde{S2}[m1]\}$ only via T21 or T23 or $\tilde{T23}$ without changing the node counter number. If T23 or $\tilde{T23}$ happens then R3 shows that node i will strictly increase n_i when leaving $\{S3[m1]\}$. If T21 $[(m1,m1)]$ happens then c) ii) shows that it cannot subsequently receive a message with $d \neq \infty$ with the same m1, and in order to enter S2[m1] again, such a message should be received. Therefore, a node can enter $\{S2[m1],\tilde{S2}[m1]\}$ at most once between $LPC(\tilde{m},m1)$ and $PC(\bar{\tilde{m}})$.

To S1[m1] a node enters only from S2[m1], so that it cannot enter S1[m1] twice between $LPC(\tilde{m},m1)$ and $PC(\bar{\tilde{m}})$.

If a node enters S3[m1], by R3 it leaves S3 only with a higher n_i, so that it cannot come back with the same n_i. q.e.d.

d) i) The assertion follows immediately from Section 3.3.3 and from the fact that the node counter number is nondecreasing, stated by Lemma A.1 b).

ii) Recall that we are always considering times before $PC(\bar{\bar{m}})$. We are going to proved) ii) for one node k such that $i \in SON_k(t)$ and the proof for all other fathers of node i follows in the same way.

Observe that

$$T21[t,i,(n1,n1]] \tag{A.43}$$

implies that $N_i(\ell)(t) = n1$ for all ℓ s.t. $F_i(\ell) = UP$. Note also that here $i \in SON_k(t)$ implies $F_i(k) = UP$, so that $N_i(k)(t-) = n1$. Note further that $D_i(k)(t-) \neq \infty$, since otherwise k was sometime before t in S3[n1] and could attach to node i only if i sent to k a message with counter number strictly higher than n1, contradicting T21[t,i,(n1,n1)].

However, $N_i(k)(t-) = n1$, $D_i(k)(t-) \neq \infty$ implies by b) that $\exists \tau \in [LPC(\tilde{m},n1),t]$ such that

$$Txy[\tau,k,SEND(n1,d,\tilde{b},i)] \quad , \quad d \neq \infty \tag{A.44}$$

Now, there are two possibilities:

If $i \notin SON_k(\tau-)$, then $Txy = T\psi 2$, but in order that $i \in SON_k(t)$, k must have performed $T21[\tau 1,k]$ at some time $\tau 1 \in (\tau,t)$.

On the other hand, if $i \in SON_k(\tau-)$, then $Txy = T21$. Therefore, in either cases, k performed:

$$T21[\eta,k,(n1,n1),(SON1,SON2)] \quad , \quad i \in SON2 \tag{A.45}$$

at some time $\eta \in [LPC(\tilde{m},n1),t]$. So $s_k(\eta+) = S1[n1]$ and $i \in SON_k(\eta+)$.

From c) and the fact that $i \in SON_k(t)$, one can easily see that k remains in S1[m1] at least until time t, q.e.d.

e) Part i) of Theorme 3.1 is trivially proved, since at any time only the SINK and nodes in state S3 have no sons.

Part ii) and iii) of Theorme 3.1, the loop-freedom property and Theorem 4.1 are proved by induction, assuming they hold up to time t- $(t \leq PC(\bar{m}))$ and showing that for any possible event at time t, these properties are preserved. To simplify the proof we look at the concatenation (n_i,s_i) and write $(n_i,s_i) \geq (n_k,s_k)$ if $n_i \geq n_k$ and if $n_i = n_k$ implies $s_i \geq s_k$. Using this notation observe from d) i) that

$$Txy[t,i,(n1,n2)] \tag{A.46}$$

implies $(n2,y) \geq (n1,x)$ for any x and y except for Txy = T21.
Note further that the induction hypothesis on 3.1 ii) and 3.1 iii) is:
If $\ell \in SON_i(t)$ then $(n_\ell,s_\ell)(t) \geq (n_i,s_i)(t) \quad \forall t \leq t-$.
Finally notice that the changes of interest here are in n_i, s_i, SON_i, p_i and d_i.

We now turn to consider all possible events at time t:

- $C\psi[t,i]$; only SON_i is changed. Since $SON_i(t) \supseteq SON_i(t+)$ the properties trivially hold at time t+.
- $\tilde{T22}[t,i]$; only s_i and possibly SON_i are changed. Since $s_i(t+) = s_i(t-)$ and $SON_i(t-) \supseteq SON_i(t+)$ the properties are preserved.
- $T\psi3[t,i]$; in these transitions $SON_i(t+) = NIL$, so that node i has no sons at time t+. Therefore, it is left to check only that the properties are preserved for fathers of node i.
  If $i \in SON_k(t-)$ then $(n_i,s_i)(t+) > (n_i,s_i)(t-) \geq (n_k,s_k)(t-)$ where the inequalities follow from Lemma A.1 b) and from the induction hypothesis respectively; so the properties are preserved for all fathers.

- T12[t,i], T22[t,1], $\tilde{T22}$[t,i]; d_i, s_i and possibly n_i are changed; no change in SON_i. Regarding fathers, the proof evolves as for Tψ3. Regarding sons, we see that

$$Txy[t,i,(n1,n2),(SON1,SON1)] \quad (A.47)$$

where Txy = T12 or T22 or $\tilde{T22}$, implies from steps B.1.1, B.1.6, B.5.1, B.5.3, B.9.1, B.9.2 in Section 3.3.3, that

$$N_i(k)(t-)=n2 \ , \ D_i(k)t-)\neq\infty \ \forall k\varepsilon SON1 \quad (A.48)$$

We now continue this part of the proof for one node $\ell\varepsilon$SON1, for each other son the proof follows in the same way.

From b) and R2, (A.48) implies that

$$\exists \tau_\ell \ \varepsilon \ [LPC(\tilde{m},n2)_-,t]$$

such that $s_\ell(\tau_\ell)$=S2[n2]. Now, if on (τ_ℓ,t), node ℓ stayed at S2[n2] or performed any transition except T21[ℓ,(n2,n2)], then the properties are preserved. Therefore, it suffices to prove that node ℓ could not have performed T21 on (τ_ℓ,t). Suppose it has , i.e.

$$T21[\tau 1_\ell,\ell,(n2,n2)] \ , \ \tau 1_\ell \varepsilon (\tau_\ell,t) \quad (A.49)$$

does happen. Then by step B.4.1 in Section 3.3.3 we have $n_\ell(i)(\tau 1_\ell)$=n2. Now we distinguish between two cases:

If $D_\ell(i)(\tau 1_\ell) \neq \infty$, then by b), $\exists\, \tau 2_\ell \in [LPC(\tilde{m},n2),\tau 1_\ell)$ such that

$$[\tau 2_\ell, i, SEND[n2,d,b,\ell)] \ , \ d \neq \infty \qquad (A.50)$$

which by property R2 implies that $s_i(\tau 2_\ell -) = S2[n2]$ or $s_i(\tau 2_\ell +) = S2[n2]$. But (A.47) says that i enters S2[n2] at time t, which contradicts c) iii).

If $D_\ell(i)(\tau 1_\ell) = \infty$, then for some time $\tau 2_\ell < \tau 1_\ell$

$$[\tau 2_\ell, i, SEND(n2,d=\infty,b,\ell)] \qquad (A.51)$$

which implies that $s_i(\tau 2_\ell +) = S3[n2]$. But $s_i(t+) = S2[n2]$ and $\tau 2_\ell < t$, which is impossible by property R3 and Lemma A.1 b). Therefore (A.49) does not happen.

- T32[t,i]; suppose

$$T32[t,i,(n1,n2),(NIL,SON2)] \qquad (A.52)$$

happens.

Regarding fathers, the properties are preserved since by property R3, n2 > n1. Regarding sons, then by b) the above implies that $\exists \tau \in (LPC(\tilde{m},n2),t]$ such that

$$[\tau, \ell, SEND(n2,d,b,i)] \ , \ \ell \in SON2 \qquad (A.53)$$

Now, from Lemma A.1 b), $n_\ell(t) \geq n_\ell(\tau)$. From property R2, $n_\ell(\tau) = n2$. Now, if $n_\ell(t) > n2$, then

$$(n_\ell,s_\ell)(t) > (n_i,s_i)(t+) = (n2,2).$$

If on the other hand, $n_\ell(t) = n2$, then the same argument as for T12, T22, $\tilde{T22}$ shows that node ℓ was in S2[n2] sometime before time t and could not return to S1[n2] in the meantime, so that

$$(n_\ell,s_\ell)(t) \geq (n_i,s_i)(t+) \qquad (A.54)$$

So, the properties are preserved in this transition.

In addition to the above, since here there is a change in SON_i from NIL to $\neq$ NIL, we have to show that a loop is not generated by this change. This is seen from the fact that every node k upstream from node i at time t has

$$(n_k,s_k](t) \leq (n_i,s_i)(t-) = (n1,3) < (n2,2) = (n_i,s_i)(t+)$$

where the first inequality follows from the induction hypothesis.

Also every node q downstream from node ℓ has

$$(n_q,s_q](t) \geq (n_\ell,s_\ell](t) > (n_i,s_i](t+) = (n2,2).$$

So, the reattachment does not generate a loop.

- T21[t,i]; suppose

$$T21[t,i,(n2,n2),(d1,d1),(SON1,SON2),(p1,p2)] \qquad (A.55)$$

happens.

Regarding fathers, i.e. if $i \in SON_k(t)$, then from d) ii) it follows that $s_k(t) = S1[n2]$, therefore

$$(n_i,s_i](t+) = (n_k,s_k](t).$$

Regarding sons, i.e. if $\ell \in SON_i(t+)$, then steps B.4.1 B.4.4, B.4.11 in Section 3.3.3 show that

$$N_i(\ell](t-) = n2,\ D_i(\ell](t-) \neq \infty\ \forall \ell \in SON_i(t+)$$

Then from b) $\exists \tau_\ell \in [LPC(m,n2),t]$ such that

$$[\tau_\ell,\ell,SEND(m,d,b,i)]\ ,\ \forall \ell \in SON_i(t+) \qquad (A.56)$$

with $m = n2 = n_\ell(\tau_\ell-)$.

Therefore, from Lemma A.1 b)

$$(n_\ell,s_\ell](t) \geq (n2,1) = (n_i,s_i](t+) \quad \forall \ell \in SON_i(t+), \qquad \text{q.e.d.}$$

This completes the proof for the properties stated in ii) and iii) in Theorem 3.1.

We now turn to prove Theorem 4.1: To prove this property notice that if (A.55) happens, then p2 ϵ $SON_i(t+)$ by steps B.4.11 in Section 3.3.3 and B.4.14 in Section 4.3.2, and if

$$(n_{p2}, s_{p2})(t+) = (n_i, s_i)(t+) = (n2, 1)$$

then by steps B.4.1, B.4.4 in Section 3.3.3 and B.4.14 in Section 4.3.2 we have

$$N_i(p2)(t-) = n2 \quad , \quad D_i(p2)(t-) \neq \infty$$

From b) $\exists \tau_{p2} \epsilon [LPC(m,n2), t]$ such that

$$[\tau_{p2}, p2, SEND(m,d,b,i)] \qquad (A.57)$$

with $m = n2$ and $d = d_{p2}(\tau_{p2}) = D_i(p2)(t-) - D'_{ip2}$.

By property R2, $s_{p2}(\tau_{p2}^{\pm}) = S2[n2]$, so by c) iii), node p2 could not enter again S2[n2] in the interval of time (τ_{p2}^{+}, t), therefore $d_{p2}(t) = d_{p2}(\tau_{p2}^{+})$. But by steps B.4.2, B.4.6, B.4.7 in Section 3.3.3 and B.4.14 in Section 4.3.2, we have

$$d_i(t+) = d1 \geq D_i(p2)(t-) = d_{p2}(t) + D'_{ip2}$$

which from assumption 2 in Section 3.3.2 implies that

$$d1 = d_i(t+) > d_{p_i}(t+) = d_{p2}(t)$$

completing the proof of Theorem 4.1, q.e.d.

In addition, to complete the proof of loop-freedom property, we have to show that the possible change in the list SON_i of sons in T21[t,i] does not generate a loop. We prove this by contradiction.

Suppose that at time t+ a loop is closed because of T21[t,i]. Since by the induction hypothesis the network was loop-free until time t-,

then the assumed loop must contain $p_i(t+)$. Denote the loop as the following string of nodes: $i_1, i_2, \ldots, i_\ell = i$, $i_1 = p_i(t+)$, i_{q+1} is a son of i_q for $q = 1,2,\ldots,\ell-1$, and i_1 is a son of i at time $t+$. Observe that at time $t-$, i_{q+1} was a son of i_q for $q = 1,2,\ldots,(\ell-1)$ too, but $i_1 \notin SON_i(t-)$ from the induction hypothesis. In addition, from Theorem 3.1 ii) and 3.1 iii) we see that around the assumed loop the concatenation (n,s) is nondecreasing, so it must be constant, namely $(n,s) = (n2,1)$ at $t+$ around the assumed loop. Clearly, this loop must contain a link (i_r, i_{r+1}) such that $d_{i_r} \leq d_{i_{r+1}}$ at time $t+$, and $(i_r, i_{r+1}) \neq (i_\ell, i_1)$ which follows from Theorem 4.1. We have already shown that $\forall \tau \in [LPC(\tilde{m},n2), t]$ such that $s_{i_1}(\tau\pm) = S2[n2]$, so by c) iii) node i_1 could not enter S2[n2] again between $\tau+$ and t. Since at time $t+$, $s_i(t+) = S1[n2]$, then $T21[\tau_{i_1}, i_1]$ happens at some time $\tau_{i_1} \in (\tau,t)$ and no other transitions happened during the interval (τ,t). Using the same arguments, we see that each node $i_q (q=1,2,\ldots,\ell)$ has been in state S2[n2] at some time after $LPC(\tilde{m},n2)$ and before t, has not entered S2[n2] again until t, and has performed $T21[i_q]$ after being in S2[n2] and before t, performing no transitions in between. While entering S2[n2], each node i_q $(q=1,2,\ldots,\ell)$ has updated its d_q and b_q, and has not done it again until t.

Let τ_{i_r}, $\tau_{i_{r+1}}$ be the last time before t nodes i_r, i_{r+1} updated their $d_{i_r}, b_{i_r}, d_{i_{r+1}}, b_{i_{r+1}}$, respectively. (From now on, slight differences appear in this part of the proof between the line switching and the message switching models. We indicate these differences when applicable).

We now claim that $d_{i_r}(t+) \leq d_{i_{r+1}}(t+)$ implies $\tau_{i_r} > \tau_{i_{r+1}}$. This is because that if $d_{i_{r+1}}$ was updated after τ_{i_r}, then it means that at τ_{i_r} node i_{r+1} was not a son of i_r, and became one on (τ_{i_r}, t), which from Theorem 4.1 and property R9 imply that $d_{i_r}(t) > d_{i_{r+1}}(t)$ contrandicting

our assumption. Therefore, at τ_{i_r}, node i_{r+1} is a son of node i_r with $d_{i_r} \leq d_{i_{r+1}}$. Clearly, for message switching, $\phi_{i_r,i_{r+1}} > 0$ and for line-switching $f_{i_r,i_{r+1}} > 0$, the latter follows from the fact that $d_{i_r} \leq d_{i_{r+1}}$ and step B.4.1.1 in Section 3.3.3. This situation is not changed at least until time t, since the only transitions nodes i_r and i_{r+1} can perform in $(\tau_{i_r},t]$ is T21, and since at time t node i_{r+1} is still a son of node i_r.

Let $t1 < t$ be the time node i_r performs T21[t1,i_r,(n2,n2)], then at t1- we have for message switching

$$na_{i_r,i_{r+1}}/t_{i_r} < \phi_{i_r,i_{r+1}} \tag{A.58}$$

for line-switching

$$na_{i_r,i_{r+1}} < f'_{i_r,i_{r+1}} \leq f_{i_r,i_{r+1}} \tag{A.59}$$

otherwise, at time t1+, $i_{r+1} \notin SON_{i_r}$ contradicting our assumption. From step B.4.8.2 in Section 3.3.3 we also have at t1-:

$$na_{i_r,i_{r+1}} \geq n[D_{i_r}(i_{r+1}) - d_{i_r}] \tag{A.60}$$

So, at t1-

for message switching

$$n[D_{i_r}(i_{r+1}) - d_{i_r}]/t_i < \phi_{i_r,i_{r+1}} \tag{A.61}$$

for line switching

$$n[D_{i_r}(i_{r+1}) - d_{i_r}] < f_{i_r,i_{r+1}} \tag{A.62}$$

However, on the interval of time $(\tau_{i_r},t1]$ all the above quantities are not changed. Therefore (A.61), (A.62) hold at τ_{i_r} also, which implies that $b_{i_r} \leftarrow 1$ at τ_{i_r} (see steps B.1.5, B.5.3, B.9.2 in Section 3.3.3).

We now notice that node i_{r-1} entered state S1[n2] only after

receiving a message MSG(n2,d≠∞,b] from node i_r, so node i_r could not become a son of i_{r-1} in T21[i_{r-1}] since i_r was blocked. Therefore node i_{r-1} set $b_{i_{r-1}} \leftarrow 1$ while entering S2[n2] (because either i_r was a son or became one in T32]. Following the same argument we move upstream on the assumed loop from i_r to i_1, and see that $b_{i_1} = 1$ at time t. But this says that i_1 was not a son of node i and became one at t+ although it was blocked at t. Step B.4.6 in Section 3.3.3 does not allow this to happen asserting a contradiction. q.e.d.

f) i) During $(\tau 1,t)$, no link is brough up by node i because of Lemma A.4. Now, suppose there were failures, let $\tau 3$ be the first time on $(\tau 1,t)$ such that

$$[\tau 3,i,FAIL(k)]; \tag{A.63}$$

Then node i performs either T23[$\tau 3$,i,(n1,n1)] or $\tilde{T}22[\tau 3,i,(n1,n1)]$ with n1 = $\bar{\bar{m}}$. In either case, d) i) shows that to exit S3[n1] or $\tilde{S}2[n1]$ and enter to S2, one has to increase n_i, so that it is impossible that

$$T21[t,i,(n1,n1)] \tag{A.64}$$

happens. So no failures can occur on $(\tau 1,t)$, q.e.d.

ii) Consider the following sequences of nodes and instants:

$$i = i_0, i_1,\ldots,i_q = SINK$$

$$t = t_0 > t_1 > t_2 > \ldots > t_q$$

such that

$$T\psi 2[t_u,i_u,(\psi,n2), (\psi,SON2_{i_u})] \tag{A.65}$$

happens, where n2 = $\bar{\bar{m}}$, $i_{u+1} \in SON2_{i_u}(t_u+)$, u = 0,1,...,q-1. Such sequences must have existed if Tψ2[t,i,(ψ,n2)] happened. Also by e) the seqeunce of nodes contains no loop until PC($\bar{\bar{m}}$).

Now, assume that $\nexists \tau_u \in [t_u, PC(\bar{\bar{m}})]$ such that

$$T21[\tau_u, i_u, (n2,n2)] \tag{A.66}$$

happens for u=0. We want to show that this assumption leads to the fact that $\nexists \tau_{u+1} \in [t_{u+1}, PC(\bar{\bar{m}})]$ such that

$$T21[\tau_{u+1}, i_{u+1}, (n2,n2)] \tag{A.67}$$

happens (remember u=0).

Suppose there existed such τ_{u+1}, then it follows from f) i) that $F_{i_{u+1}}(i_u)(\tau_{u+1}) = UP$ (since $F_{i_{u+1}}(i_u)(t_{u+1}) = UP$).

The next step of this proof is to show that $N_{i_{u+1}}(i_u)(\tau_{u+1}-) \neq \bar{\bar{m}} = n2$. To do this we must show that $\nexists \tau 2_{u+1} < \tau_{u+1}$ such that

$$[\tau 2_{u+1}, i_u, SEND(n2,d,b,i_{u+1})], d = \infty \tag{A.68}$$

and that $\nexists \tau 3_{u+1} \in [PC(\bar{\bar{m}}), \tau_{u+1})$ such that

$$[\tau 3_{u+1}, i_u, SEND(n2,d,b,i_{u+1})] \ , \ d \neq \infty \tag{A.69}$$

For $\tau 2_{u+1} < t_u$, it follows that $\nexists$ such $\tau 2_{u+1}$ from properties R2 and R3 (since $s_{i_u}(\tau 2_{u+1}^+) = S3[n2]$ and it can't be that $s_{i_u}(t_u) = S2[n2]$). For $\tau 3_{u+1} < t_u$, it follows from property R2 and c) iii) (since $s_{i_u}(\tau 3_{u+1}^+) = S2[n2]$ and it can't enter S2[n2] again at t_u again).

For $\tau 2_{u+1} = t_u$ or $\tau 3_{u+1} = t_u$, it follows from the fact that $i_{u+1} \in SON2_{i_u}(t_u)$ and i_u does not send a message to its sons in TΨ2.

For $\tau 2_{u+1} \in (t_u, PC(\bar{\bar{m}}))$ or $\tau 3_{u+1} \in (t_u, PC(\bar{\bar{m}}))$, the only possibilities for i_u if $T21[i_u]$ does not happen are: to stay in $S2[\bar{\bar{m}}]$ or $\tilde{T22}[(n2,n2)]$ or $\tilde{T23}[(n2,n2)]$ or $T23[(n2,n2)]$ or $\tilde{C2}[(n2,n2)]$. In all cases, i_u will not send any message to i_{u+1}.

The above implies that $N_{i_{u+1}}(i_u)(\tau_{u+1}-) \neq \bar{\bar{m}} = n2$ so that

$$T21[\tau_{u+1}, i_{u+1}, (n2,n2)] \tag{A.70}$$

is impossible (by step B.4.1 in Section 3.3.3)

Regarding the proof by induction (increasing u by 1) shows that

$\nexists \tau_q$ such that

$$T21[\tau_q, SINK, (n2,n2)] \ , \ n2 = \bar{\bar{m}} \tag{A.71}$$

happens, which contradicts the assumption that there is a proper completion at time $PC(\bar{\bar{m}})$.

This proves the first part of f) ii). The second part of f) ii) follows because $T21[\tau 1,i,(n2,n2)]$, $n2 = \bar{\bar{m}}$ is not possible if $T32[\tau 2,i,(n2,n2)]$ or $\tilde{T22}[\tau 2,i,(n2,n2)]$ happen, q.e.d.

g) If

$$[\tau,k,SEND(\bar{\bar{m}},d,b,i)] \ , \ d \neq \infty \tag{A.72}$$

then $F_k(i)(\tau) = UP$ and by property R2 either

$$T\psi 2[\tau,k,(\psi,n2)] \ , \ n2 = \bar{\bar{m}} \tag{A.73}$$

or

$$T21[\tau 1,k,(n2,n2)] \ , \ n2 = \bar{\bar{m}}. \tag{A.74}$$

If $T\psi 2$ then f) ii) implies that $\exists \tau 2 \in (\tau, PC(\bar{\bar{m}}))$ such that

$$T21[\tau 2,k,(n2,n2)] \ , \ n2 = \bar{\bar{m}} \tag{A.75}$$

and $F_k(i)(\tau 2) = UP$. Therefore T21 happens at node k at some time ($\tau 1$ or $\tau 2$). Call this time η. We have then $N_k(i)(\eta) = \bar{\bar{m}}$. By b) either $\exists \tau 3 \in [PC(\bar{\bar{m}}), \eta]$ such that

$$[\tau 3,i,SEND(\bar{\bar{m}},d,b,k)] \ , \ d \neq \infty \tag{A.76}$$

or $\tau 4 < \eta$ such that

$$[\tau 4,i,SEND(\bar{\bar{m}},d,b,k)] \ , \ d = \infty \tag{A.77}$$

but by property R2, this means that node i is at $\tau 4 < PC(\bar{\bar{m}})$ in $S3[\bar{\bar{m}}]$ or is at $PC(\bar{m}) < \tau 3 < PC(\bar{\bar{m}})$ in $S2[\bar{\bar{m}}]$.

If the first holds, node i will stay in $S3[\bar{\bar{m}}]$ at least until $PC(\bar{\bar{m}})$. If the latter holds, then by f) ii) it must perform $T21[i,(n2,n2)]$, $n2 = \bar{\bar{m}}$ before $PC(\bar{\bar{m}})$. But since at time t it still has $N_i(k)(t) = \bar{\bar{m}}$, $D_i(k)(t) \neq \infty$ or has not received yet the message by time t, c) i) implies that node i could not perform $T21[i,(n2,n2)]$, $n2 = \bar{\bar{m}}$ before time t. Therefore it will perform it later, q.e.d.

Proof that a) holds at time $PC(\bar{\bar{m}})$:

i) Node i cannot be in $S2[\bar{\bar{m}}]$ because of f) ii) and c) iii). It can't be in $\tilde{S2}[\bar{\bar{m}}]$ because it must have been in $S2[\bar{\bar{m}}]$ before and because of f) ii).

ii) Take $t = PC(\bar{\bar{m}})$ in g). It follows then that $s_i(PC(\bar{\bar{m}})) = S2[\bar{\bar{m}}]$ or $S3[\bar{\bar{m}}]$ but f) ii) and c) iii) imply that $s_i(PC(\bar{\bar{m}})) \neq S2[\bar{\bar{m}}]$.

iii) Follows by contradiction, since if at $PC(\bar{\bar{m}})$

$$N_i(k)(PC(\bar{\bar{m}})) = \bar{\bar{m}} \ , \ D_i(k)(PC(\bar{\bar{m}})) \neq \infty$$

it follows by taking $t = PC(\bar{\bar{m}})$ in g) that $s_i(PC(\bar{\bar{m}})) = S2[\bar{\bar{m}}]$ or $S3[\bar{\bar{m}}]$, q.e.d.

This completes the proof of Theorem A.1.

APPENDIX B

In appendix A we have proved Theorems 3.1, 3.2, 3.3 and 4.1. This appendix is devoted to proofs of the remaining statements of our work, namely Theorems 3.4, 3.5 and 4.2. The proofs are organized as follows: Lemma B.0 is preliminary and shows that on any link (i,ℓ) the only two "stable situations are $\{F_i(\ell) = F_\ell(i) = \text{DOWN}\}$ or $\{F_i(\ell) \neq \text{DOWN}, F_\ell(i) \neq \text{DOWN}\}$. Lemmas B.1 and B.2 prove Theorems 3.5, Lemma B.3 proves Theorems 3.4 and Theorem 4.2 is proved by the series of Lemmas - B.4 to B.7.

Lemma B.0

If $F_i(\ell)(t1) = \text{DOWN}$, $F_\ell(i)(t1) \neq \text{DOWN}$, then in finite time after t1 we have either $\{F_i(\ell) = F_\ell(i) = \text{DOWN}\}$ or $\{F_i(\ell) \neq \text{DOWN}, F_\ell(i) \neq \text{DOWN}\}$.

Proof

$F_\ell(i)(t1) \neq \text{DOWN}$ means either $F_\ell(i)(t1) = \text{READY}$ or $F_\ell(i)(t1) = \text{UP}$. If $F_\ell(i)(t1) = \text{READY}$, then nodes i and ℓ arrived at this situation from $\{F_i(\ell) = F_\ell(i) = \text{DOWN}\}$ or $\{F_i(\ell) = F_\ell(i) = \text{READY}\}$ or $\{F_i(\ell) = \text{UP}, F_\ell(i) = \text{READY}\}$. Then assumptions 9 in Section 3.3.2 imply the assertion.

If $F_\ell(i)(t1) = \text{UP}$, then nodes i and ℓ arrived at this situation from $\{F_i(\ell) = \text{DOWN}, F_\ell(i) = \text{READY}\}$ or $\{F_i(\ell) = F_\ell(i) = \text{UP}\}$ or $\{F_i(\ell) = \text{READY}, F_\ell(i) = \text{UP}\}$. In the first case, the discussion reduces to the first part of the proof, whereas for the second and the third cases, assumption 9.a) in Section 3.3.2 proves the assertion.

Lemma B.1

Theorem 3.5 a).

Proof

Clearly, $n_i(t1-) < m2$ for all i (property R8). Therefore (3.16) may happen only at or after t1. Let us now define four sets of nodes:

$A(t) = \{i \mid i \in L(t)$ and i effected (3.16) with $t2_1 < t\}$,

$B(t) = \{i \mid i \in L(t)$ and $i \notin A(t)\}$,

$A'(t) = \{i \mid i \in A(t)$ and i has a potentially working link to a node in $B(t)\}$,

$B'(t) = \{i \mid i \in B(t)$ and i has a potentially working link to a node in $A(t)\}$.

If there is an instant t2 such that $A(t2) = L(t2)$, then the proof is complete. Otherwise, for a given instant t3, we will show (by contradiction) that there is an instant t, $t3 < t < \infty$ such that

$$A(t) \supseteq A(t3) \quad \text{and} \quad A(t) \neq A(t3). \tag{B.1}$$

Hence by induction, the set A(t) keeps growing until it equals L(t).

Since there are no pertinent topological changes and since all nodes $i \in A(t)$ have $n_i(t) = m2$, property R11 implies that the set A(t) is non-decreasing as t increases. Therefore, to proved part i) of Theorem 3.5 a) it is sufficient to show that the following cannot hold:

$$\forall t > t3 \ , \ A(t) = A(t3) \neq L(t) \tag{B.2}$$

We contradict (B.2) by the following three claims:

CLAIM 1

If (B.2) holds, then $\exists\, t4 \in (t3,\infty)$ such that $\forall\, j \in B'(t4)$, $\exists\, t4_{jk} < t4$ such that

$$[t4_{jk}, j, MSG(m2, d\neq\infty, k)], \tag{B.3}$$

$\forall\, k$ such that $k \in A'(t4)$ and $F_k(j)(t3) = UP$ (i.e. all nodes of $B'(t4)$ receive m2 in finite time from all their neighbors in $A'(t4)$).

Proof of CLAIM 1

At time $t2_i < t3$, node $i \in A'(t2_i)$ performs (3.16). For links (i,ℓ), where $i \in A'(t2_i)$, $\ell \in B'(t2_i)$ and $F_i(\ell)(t2_i+) = UP$, observe from steps B.1.8, B.5.3, B.8.9, B.9.2 in Section 3.3.3 that if $\ell \notin SON_i(t2_i)$, then

$$[t2_i, i, SEND(m2, d\neq\infty, \ell)]. \tag{B.4}$$

Notice further that for each node $k \in SON_i(t2_i)$ we know from Theorem A.1 e) that $k \notin B(t2_i)$. Observe also that since no pertinent topological changes occur, property R11 insures that for all nodes ℓ, $F_i(\ell)$ cannot be changed from or to DOWN after $t2_i$ for $i \in A'(t2_i)$. It means that if $F_i(\ell)(t2_i-) = DOWN$ then $F_i(\ell)(t) = DOWN\ \forall t \geq t2_i$ and if $F_i(\ell)(t2_i-) = READY$ or UP, then $F_i(\ell)(t) = UP\ \forall t \geq t2_i$ (see steps B.1.8, B.5.3, B.8.8, B.9.2 in Section 3.3.3). Therefore, from assumption 3 in Section 3.3.2 it is insured that the message in (B.4) arrives at ℓ in finite time. So, there is a time t4 for which all nodes j that were in $B'(t2_k)$ for some k, either are not in $B'(t4)$ anymore or have received $MSG(m2, d\neq\infty, k)$ at time $t4_{jk} < t4$ from all nodes k such that $k \in A'(t4)$ and $F_k(j)(t3) = UP$.

q.e.d. CLAIM 1.

Notice now that B'(t4) cannot be emplty, since then (B.2) is contradicted.

Let $t5_j = \max_k(t4_{jk})$ for a node $j \in B'(t4)$ where $t4_{jk}$ is as defined in CLAIM 1. There exists such time $t5_j < \infty$ because of Claim 1 and since there is a finite number of nodes in the network, (in words, $t5_j$ is the time that a node $j \in B'(t4)$ has received MSG(m2,d≠∞] from all its neighbors in A'(t4)).

Now, if $\exists j \in B'(t4)$ such that $\forall k \in SON_j(t5_j)$, $k \in A'(t4)$, then from steps B.1.1, B.1.2, B.5.1, B.5.2, B.9.1 in Section 3.3.3.

$$T\psi 2[t5_j, j, (\psi, m2)] \tag{B.5}$$

happens, contradicting (B.2) q.e.d.

Therefore, we now assume that $\forall j \in B'(t4)$ the following cannot hold: $\{\forall k \in SON_j(t5_j)\ ,\ k \in A'(t4)\}$.

CLAIM 2

If $j \in B'(t4)$ and $\{\forall k \in SON_j(t5_j)\ ,\ k \in A'(t4)\}$ does not hold, then $\forall t > t4_j$ the following cannot hold: $\{\forall k \in SON_j(t),\ k \in A'(t4)\}$.

Proof of CLAIM 2

Suppose there is time $t > t5_j$ such that $\{\forall k \in SON_j(t),\ k \in A'(t4)\}$. Then, for the first time after $t5_j$ it holds

$$Txy[t, j, (SON1, SON2)]\ ,\ t > t5_j \tag{B.6}$$

or $\quad Cx[t,j,(SON1,SON2)] \quad , \quad t > t5_j \qquad (B.7)$

happens with SON1 ≠ SON2.

- If T22 or T12, then SON1 = SON2, so these transitions do not hold here.
- If Tψ3, then SON2 = NIL ≠ k, hence cannot happen.
- If T21, then $\forall q$, $N_j(q) = n_j < m2$, but $N_j(k)(t] = m2$, hence T21 cannot happen.
- If T32, then SON1 = NIL and T32[t,j,(ψ,m2)] happens, contradicting (B.2), hence cannot happen.
- If $\tilde{T22}$ or C1 or $\tilde{C2}$, then exactly one node is delected from SON1, call it ℓ. After this node is deleted, we assumed that $\forall$ k ε SON2, then k ε A'(t3), therefore node j will effect $\tilde{T22}$[t,j,(ψ,m2)] or T12[t,j,(ψ,m2)] since $n_j < m2$, which contradicts (B.2), hence cannot happen.

q.e.d. CLAIM 2.

CLAIM 3

In finite time, all nodes i ε B(t4) will effect Tψ3[i,(ψ,m)] with m < m2, without effecting T32 thereafter.

Proof of CLAIM 3

We know from Section 3.3.3 that n_i can be updated only in transitions Tψ2 and Tψ3. For all nodes i ε B(t4), Tψ2[i,(ψ,m2)] does not occur, otherwise (B.2) is contradicted. Also Tψ3[i,(ψ,m2)] does not occur for all nodes i ε B(t4) because no message of the type MSG(m2,d=∞) is generated in the network since there are no pertinent topological changes. Therefore,

$\forall i \in B(t4)$ and $\forall t > t4$, then $n_i(t) < m2$. (B.8)

After time t4, no update iteration with m < m2 is started by the SINK (since the SINK has started an iteration with m = m2 before t4). By Theorem 3.2 ii) it implies that the number of messages with $d \neq \infty$ generated by the nodes that belong to B(t4) is finite (remember we deal with a finite number of nodes in the network). Similarly, since the number of links is finite, the number of FAIL messages is also finite. Let t'_{mx} be the time after all these messages are generated and received. Define $t_{mx} = \max(t4, t'_{mx})$. Clearly, $\forall i \in B(t4)$, T32[i] cannot occur after t_{mx} (since all FAIL messages and MSG($d\neq\infty$) have been already received).

We now define the following set of nodes:

$\overline{B}(t) = \{i \mid i \in B(t4) \text{ and } SON_i(t) = NIL\}$.

If $\overline{B}(t_{mx}) = B(t4)$ then q.e.d. claim 3. Otherwise, there are nodes $k \in B(t4)$ and $k \notin \overline{B}(t_{mx})$. All these nodes, after a sufficiently long period of time - t^*_{mx}, will not have sons which belong to $\overline{B}(t_{mx})$ (since nodes in $\overline{B}(t_{mx})$ effect Txy[i,SEND(m,d=∞)] when SON_i is set to NIL, therefore they are deleted from the list of sons of every $k \in B(t4)$ and $k \notin \overline{B}(t_{mx})$). Since there are no loops in the network, at t^*_{mx} there is a node $i \in B(t4)$ and $i \notin \overline{B}(t_{mx})$ which has no son also in the set of nodes $\{k \mid k \in B(t4) \text{ and } k \notin \overline{B}(t_{mx})\}$. By CLAIM 2, this node neither has all its sons in A'(t4). Consequently, at t^*_{mx}, this node has no sons at all, so $s_i(t^*_{mx}) = S3$ and it cannot leave S3 thereafter (since all messages MSG($d\neq\infty$) has already been received). By induction, the set of nodes $\overline{B}(t)$ grows until it equals B(t4).

q.e.d. CLAIM 3.

The proof of Theorem 3.5 a) i) is completed as follows:

Consider a node $j \in B'(t4)$. Define $t3_j$ to be the time at which $T\psi3[t3_j,j]$ occurs by CLAIM 3. But,

if $t3_j < t5_j$, then $T32[t5_j,j]$ happens,

if $t3_j > t5_j$, then $T32[t3_j,j]$ happens,

and $t3_j \neq 5t_j$ since j processes the messages one at the time. This contradicts (B.2). So by induction, the set A(t) keeps growing until it equals L(t).

q.e.d.

To prove part (ii) of Theorem 3.5 a), we investigate further the situation in L(t2) at time t2. Observe that since all nodes $i \in L(t2)$ have $n_i = m2$, and since no pertinent topological changes occur, it follows from R11 and Lemma B.0 that for any link (i,ℓ) such that $i \in L(t2)$ and $\ell \in L(t2)$, it <u>cannot</u> happen that at time t2 we have $F_i(\ell) = DOWN$, $F_\ell(i) \neq DOWN$. Also $F_i(\ell) = READY$ is impossible, because lack of pertinent topological changes imply that $F_i(\ell)(t2_i-) = READY$ as well, and then by steps B.1.8, B.5.4, B.8.8, B.9.2 in Section 3.3.3 $F_i(\ell)(t2_i+) = UP$, therefore $F_i(\ell)(t2) = UP$. Consequently, for links (i,ℓ) connecting nodes in L(t2), the only possibilities <u>at time t2</u> are $\{F_i(\ell) = F_\ell(i) = DOWN\}$ or $\{F_i(\ell) = F_\ell(i) = UP\}$, hence part ii) of Theorem 3.5 a) is proved.

Next assuming Theorem 3.5 a)which was proved by Lemma B.1, we now prove Theorem 3.5 b).

Lemma B.2

Theorem 3.5 b).

Proof

We first prove part i) of Theorem 3.5 b) by showing that there is PC(m1) after t1 and that there is no PC(m1) between t1 and t2.

Lack of pertinent topological changes insures that after entering S2[m1] at $t2_i$, each node $i \in L(t2)$ can only perform transitions between states S1[m1] and S2[m2]. Furthermore, by Theorem 3.1 i) notice that after t2, these nodes form a loop-free pattern (lattice) with the SINK the only terminating node. Consider a time t', t' > t2. L(t') = L(t2) since there are no topological changes. Also, by Theorem 3.2 iii), if a node $i \in L(t2)$ enters S2[m1] after t2, then PC(m1) has occured after t1.

1. If $s_i(t') = S1[m1]$, $i \in L(t')$, then there exists t3, t1 < t3 < t' such that T21[t3,SINK,(m1,m1)] happened (since SINK $\in$ L(t'));

2. Otherwise, consider a node $k \in L(t')$ such that $s_k(t') = S2[m1]$ and $\forall j$, if $k \in SON_j(t')$, then $s_j(t') = S1[m1]$ (notice there can exist no such a node j). Such a node k must exist if not all the nodes are in S1[m1]. Classify the neighbors of k into the two following sets of nodes:

$$A = \{i \mid F_i(k)(t') = UP \text{ and } s_i(t') = S1[m1]\},$$

$$B = \{i \mid F_i(k)(t') = UP \text{ and } s_i(t') = S2[m1]\}.$$

At some time in the interval [t1,t'], each node $i \in A$ has sent messages

MSG(m1,d≠∞) to all its neighbors, namely to all nodes q such that $F_i(q)(t')$ = UP. Also, at some time during the same interval, each node i ε B has sent such messages to all their neighbors except sons, namely to all nodes q such that $F_i(q)(t')$ = UP and q ∉ $SON_i(t')$. However, k is not a son of any of those nodes in B (since it is a son only of nodes in A). Hence,by 3 in Section 3.3.2 node k will receive messages MSG(m1,d≠∞) from all its neighbors, at a finite time, say t4. Then:

2.1: If $s_k(t4+)$ = S2[m1], then ∃i with $F_k(i)(t4)$ = UP such that $N_k(i)(t4)$ = NIL, which implies that T21[k,(m1,m1)] happened in the interval [t1,t4], hence by Theorem 3.2 iii), PC(m1) occured between t1 and t4.

2.2: If $s_k(t4+)$ = S1[m1], then by induction PC(m1) will happen in finite time after t1.

We show next that PC(m1) cannot happen in [t1,t2]. Suppose t5 is the first time PC(m1) occurs after t1 and t5 < t2. It means there is a node k ε L(t2) such that $t2_k$ > t5. Also there exists a node j ε L(t2) such that $F_j(k)(t2_j)$ = UP and since there are no pertinent topological changes $F_j(k)(t5)$ = UP too. So, node j has sent to k a message MSG(m1,d≠∞) in the interval $[t2_j,t5]$. If at time t5 the message is on its way to k, then by Theorem 3.3 ii), s_k = S3[m1] which contradicts the lack of pertinent topological changes. If the message has arrived to k before t5, then at time t5 either n_k < m1 or s_k = S1[m1] (since k has not entered S2[m2] yet), and by Theorem 3.3 iii) for all nodes i such that $F_k(i)$ = UP, including j, it <u>cannot</u> happen that

$\{N_k(i) = m1,\ D_k(i) \neq \infty\}$. Therefore, such a node k does not exist, which means that t5 > t2.

q.e.d. i)

Furthermore, we notice that since there are no pertinent topological changes, we have L(t2) = L(t3), and according to Theorem 3.1 i) the Routing Graph of these nodes forms a loop-free pattern with the SINK the only terminating node.

q.e.d. iii)

Finally, looking at the situation in the network at time t2 as described in Lemma B.1, and for all t ε [t2,t3], we observe that for all links (i,ℓ) for which $F_i(\ell)(t2)$ = UP we must have $F_i(\ell)(t)$ = UP and if $F_i(\ell)(t2)$ = DOWN, then we must have $F_i(\ell)(t)$ = DOWN. This completes the proof of ii).

q.e.d. Theorem 3.5 b).

Lemma B.3

Theorem 3.4

Proof

From Section 4.3.2 we know that a new iteration T21[t1,SINK,(m1,m1)] can start only if all previous iterations with the same counter number m1 were properly completed. Since iteration counter numbers are nondecreasing, the first iteration with m1 has been started at some time, say t', by

$$T12[t',SINK,(m0,m1)] \quad , \quad m1 > m0. \tag{B.9}$$

This transition satisfies the conditions of Theorem 3.5. Hence in a finite time, say t0, the iteration is properly completed, and from Section 3.3.3 t0 > t1, q.e.d. Theorem 3.4 a). Also L(t0) forms a loop-free pattern (lattice) with the SINK the only terminating node, and $n_i = m1$, $i \in L(t0)$, and since there are no pertinent topological changes, for all $t \geq t0$ we have:

1. H(t) = L(t) = L(t0) q.e.d. Theorem 3.4 b).

2. By Theorem 3.1 i) all nodes $i \in L(t)$ form a lattice with the SINK the only terminating node, q.e.d. Theorem 3.4.d ii).

We prove Theorem 3.4 c) by induction: First, we notice that since there are no pertinent topological changes, then all nodes $i \in L(t1)$ can perform only transitions between S1[m1] and S2[m1].

We now define two sets of nodes:

$$A(t) = \{i \mid i \in L(t) \text{ and } i \text{ effected } T12[t2_i,i,(m1,m1)], \text{ with } t1 \leq t2_i \leq t\};$$

$$B(t) = \{i \mid i \in L(t) \text{ and } i \notin A(t)\}.$$

The induction is done over the set A(t) and we want to show that it grows until it equals L(t). Clearly A(t1) = SINK. Assume the set $A(t_r)$ contains several nodes for $t_r \geq t1$. Take a node $k \in B(t_r)$ which all its sons belong to $A(t_r)$ at t_r (there must exist such a node since the network is loop-free). Node k can change its list of sons only via T21, so it does not change this list at least until k enters S2[m1]. At t_r^+ all nodes in $A(t_r)$ have sent messages MSG(m1,d$\neq\infty$) to all their

neighbors, except possibly their sons. However, node k cannot be a son of any of the nodes in $A(t_r)$ (since the network is loop-free). By 3 in Section 3.3.3 node k receives messages MSG(m1,d≠∞) from all its sons in finite time, say $t2_k$. Therefore at $t2_k$ node k performs (see step B.1.1 in Section 3.3.3).

$$T12[t2_k, k, (m1, m1)]. \tag{B.10}$$

We now can add node k to $A(t2_k)$ and delete it from $B(t2_k)$. By induction, the set $A(t)$ keeps growing until it equals L(t), q.e.d. Theorem 3.4 c).

Theorem 3.4. d)i) follows directly from Lemma B.2 by assuming Theorem 3.4. c). q.e.d. Theorem 3.4.

Theorem 4.2 will be proved by Lemmas B.4, B.5, B.6 and B.7. Lemma B.4 is preliminary and is used to simplify the following proofs. Lemma B.5 deals with the case when a node in S2 or $\tilde{S2}$ sends a REQ(m1) message. Lemma B.6 proves the Theorem for the case where there is a node in state S3[m1]. Lemma B.7 is similar to Lemma B.5 but for S1.

Lemma B.4

If a REQ(m1) is generated in the network, then either

1. all nodes j in the entwork have $n_j \leq m1$ and REQ(m1) is processed only by nodes having $n_i = m1$

   or

2. a REQ(m1) arrived at the SINK.

Proof

By Theorem 3.1 ii) and from Section 4.3.2, REQ(ml) is not received by a node i with $n_i < ml$. On the other hand, if there exists a node i with $n_i > ml$, the SINK started an iteration with $m > ml$; this is equivalent to the arrival of REQ(ml) to the SINK, q.e.d.

Lemma B.5

If a node i sends REQ(ml) while $s_i = S2[ml]$ or $s_i = \tilde{S2}[ml]$, then a REQ(ml) arrived or will arrive at the SINK in finite time.

Proof

Consider the following sets of nodes and intervals of time:

$$i = i_o \in A_o, A_1, A_2, \ldots, A_s = SINK$$

$$TIM_o > TIM_1 > TIM_2 > \ldots > TIM_s$$

such that

$$T\psi 2[t_{i_r}, i_r, (\psi, n2), (\psi, SON2_{i_r}), (\psi, p2_{i_r})] \qquad B.11)$$

happens, where $n2 = ml$, $t_{i_r} \in TIM_r$, $i_r \in A_r$ and $SON2_{i_r} \subset \bigcup_{\alpha=r+1}^{s} A_\alpha$, $p2_{i_r} \in SON2_{i_r}$, $r=0,1,\ldots,s$. Such sets of nodes and intervals of time must exist if $s_i = S2[ml]$ or $s_i = \tilde{S2}[ml]$. There is no loop in the set of nodes $\bigcup_{\alpha=0}^{s} A_\alpha$ at all times, otherwise Lemma B.4, Theorems 3.1,3.2 or 3.3 will be contradicted.

The proof proceeds by induction. We know that i_o sends REQ(ml) to $p2_{i_o}$ (unless it has lost it) while being in $S2[ml]$ or $\tilde{S2}[ml]$.

Set r=0. Suppose that at time $t2_{i_r} > t_{i_r}$, node i_r sends REQ(m1) to its preferred son $i_q = p2_{i_r} \in A_q$ with $q \geq r+1$. The case when $p2_{i_r} = NIL$ is discussed later. Suppose also that during the interval of time $[t_{i_r}, t2_{i_r}]$ node i_r performs no transition except possibly $\tilde{T22}$ or $\tilde{C2}$. Then after t_{i_q}, the first transition executed by node i_q could be:

- $T22[i_q]$: q.e.d. by Lemma B.4 (since in T22 node i_q strictly increases its node counter number from m1).
- $\tilde{T22}[i_q, FAIL(\ell)]$: then node i_q detects a failure and sends REQ(m1) to its preferred son while being in $\tilde{S2}[m1]$.
- $T21[i_q]$: this transition is executed only after receiving a message from i_r. Such a message is sent by i_r when $T21[i_r]$ occurs, i.e. after i_r has sent the REQ(m1). FIFO at node i_q shows that i_q will receive and therefore send REQ(m1) to its perferred son before $T21[i_q]$ occurs, i.e. while $s_{i_q} = S2[m1]$.
- $T23[i_q]$ or $\tilde{T22}[i_q, MSG(d=\infty)]$: in this case there exists a node $i_\ell \in A_\ell$, $\ell > q$ such that $\tilde{T22}[i_\ell, FAIL]$ occurs and i_ℓ sends REQ(m1) to its preferred son while being in $\tilde{S2}[m1]$.

If i_q performs no transition, then it sends REQ(m1) to its preferred son while $s_{i_q} = S2[m1]$.

Thus, by induction, increase r by q, a string of nodes is established, in which each node sends REQ(m1) to its preferred son, and if for each node i in the string, $p_i \neq NIL$ then REQ(m1) arrived or will arrive at the SINK. Finally we check the possible case that one (or more) of the nodes of the string described above has $p_i = NIL$. In such a case, since while entering S2[m1] each node in this string has $p_i = NIL$ (as determined by

step B.1.10, B.5.3, B.8.12, B.9.2 in Sections 3.3.3 and 4.3.2) then i lost its p_i after all nodes downstream from it have entered S2[m1]. Therefore, there exists a downstream node q which detects a failure and $\tilde{T22}$[q,FAIL] happens at that node (since the network is loop-free) and it sends REQ(m1) while being in S2[m1] or $\tilde{S2}$[m1]. Induction asserts that REQ(m1) arrived or will arrive at the SINK.

q.e.d.

Lemma B.6

If there exists a node that performs $T\psi 3[(\psi,m1)]$, then a REQ(m1) arrived or will arrive at the SINK in finite time.

Proof

Let PC_j, j = 1, 2, ... denotes the j-th occurence of PC(m1). Given a node i and a time t such that $T\psi 2[i,(\psi,m1)]$ has happened before t, if PC_j is the last PC(m1) before t, afterwhich $T\psi 2[i,(\psi,m1)]$ happened, then define $E_i(t) = j$.

Property

Given a time t, suppose that $k \in SON_i(t)$ and $n_i(t) = n_k(t) = m1$, then $E_i(t) \leq E_k(t)$.

Proof of the Property

Let t1 be the time, node k was last set to be a son of node i before time t. This can be done only via T21[i] or T32[i]. Let PC_j be the last

proper completion before t1. By Theorem 3.3 $s_i[PC_j] \neq S2[m1]$, therefore $T\psi2[i,(\psi,m1)]$ happened after PC_j and it cannot happen again before t because of Theorem 3.2 iii].

Hence $E_i(t1) = j$

The occurence of T21[i] or T32[i] implies that a message MSG($d \neq \infty$) has been received at i from k after PC_j. By Theorem A.1 b] this message was sent by k after PC_j and this can only be done if k performed $T\psi2[k]$ after PC_j. Since E_k is a nondecreasing number, then $E_k(t1) \geq j$. Since at time t, k still belongs to $SON_i(t)$, then node i cannot enter state S2 on the interval (t1,t) unless node k has entered S2, implying that $E_i(t) \leq E_k(t)$.

q.e.d. (the property).

We may now continue the proof:

By Lemma B.4 we have to prove this Lemma only for the case in which for all nodes in the network we have $n_i \leq m1$. Therefore, a node that effects $T\psi3[(\psi,m1)]$ cannot effect any more transitions (by property R3). Since the number of links in the network is finite, then only a finite number of transitions $T\psi3[(\psi,m1)]$ can be executed. If $T\psi3[(\psi,m1)]$ happens, then there exists a node which detects a failure on its link to its only son and executed $T\psi3[(m1,m1)]$ (see steps B.2.1 - B.2.3, B.2.7, B.7.3, B.10.3 in Section 3.3.3). Define B1 as the set of nodes for which $T\psi3[(m1,m1)]$ happens, namely

$B1 = \{i \mid T\psi3[t_i,i,(m1,m1)] \text{ happens}\}$.

Define B2 as a subset of nodes of B1 with the highest E_j, namely

$$B2 = \{\ell \mid \ell \in B1 \text{ and } E_\ell(t_\ell) = \max_{i \in B1} E_i(t_i)\}.$$

CASE 1: Suppose there exists a node $\ell \in B2$ that effects

$$T23[t_\ell, \ell, (m1, m1)] \text{ or } \tilde{T23}[t_\ell, \ell, (m1, m1)] \tag{B.12}$$

Let $\max_{i \in B2} E_i(t_i) = j$. Then at PC_j (by Theorem 3.3) $s_i(PC_j) \neq S2[m1]$. Thus the first node $\ell \in B2$ that effects (B.12) has at least one route to SINK at t_ℓ (by Theorem 3.1 i)). From all nodes $\ell \in B2$ that effects (B.12) while having a route to SINK at t_ℓ, let us choose a node q_o such that $q_1 \in SON_{q_o}(t_{q_o})$ and $q_1 \notin B2$. (there must exist such a node since SINK $\notin$ B2). Because of the property proved above $q_1 \notin B1$, so q_1 does not enter S3. Also by Theorem 3.1, $s_{q_1}(t_{q_o}) = S2[m1]$ or $\tilde{S2}[m1]$ and q_1 can only effect $\tilde{T22}$ or $\tilde{C2}$, because we have shown it cannot effect $T\psi 3$, and it cannot effect also T21 unless it receives a message $MSG(d \neq \infty)$ from q_o, which cannot be sent since q_o does not effect T21.

Hence, q_1 will detect a failure at link (q_o, q_1) and send REQ(m1) while $s_{q_1} = S2[m1]$ or $\tilde{S2}[m1]$, and by Lemma B.5 the assertion follows.

CASE 2: Suppose there exists no node $\ell \in B2$ that effects (B.12), i.e. every node $\ell \in B2$ effects $T13[t_\ell, \ell, (m1, m1)]$. Let $q_o \in B2$ denotes a node such that

$$d_{q_o}(t_{q_o}) = \min_{i \in B2} \{d_i(t_i-)\} \tag{B.13}$$

When q_o entered S1[m1] at the last time before t_{q_o}, it had its preferred son p_{q_o} (see step B.4.14 in Section 4.3.2.)

p_{q_o} cannot effect T23 because this will violate CASE 2,and cannot effect T13 because this violates either (B.13) or CASE 2. Therefore p_{q_o} detects a failure on link (p_{q_o}, q_o) and sends REQ(m1).

If at any time this REQ(m1) is processed by a node at state S2 or state $\tilde{S2}$, then the assertion follows by Lemma B.5. Otherwise the REQ(m1) keeps moving through nodes at S1[m1] since it cannot be received by a node at state S3 because this violates either CASE 2 or (B.13). The REQ(m1) is forwarded from each node to its preferred son, thus it moves through nodes having decreasing d's by Theorem 4.1. Even if the REQ(m1) arrives at a node, ℓ say, with p_ℓ= NIL, then node ℓ detected a failure on link (ℓ, p_ℓ) and p_ℓ has sent REQ(m1) and by Theorem 4.1 $d_{p_\ell} < d_\ell$ when it has been sent.

Since for all nodes i, $d_i \geq 0$, d_i is an integral number and the only node with $d_i = 0$ is the SINK, the REQ(m1) will arrive at SINK after a finite number of steps, q.e.d.

Lemma B.7

If a node i sends a REQ(m1) while s_i = S1[m1] then a REQ(m1) arrived or will arrive at SINK in finite time.

Proof

If there exists a node ℓ such that s_ℓ = S3[m1] then q.e.d. by Lemma B.6. Hence we may assume that for all nodes ℓ in the network $s_\ell \neq$ S3[m1]. Also, by Lemma B.4 we know that the REQ(m1) sent by i may

encounter only nodes having n_i = m1. Thus as in the proof of Lemma B.6 the REQ(m1) either arrives at a node S2 or S2 (q.e.d. by Lemma B.5) or moves through nodes at S1, with decreasing d's until it arrives at SINK, q.e.d.

APPENDIX C

In this appendix we give the program that simulates the operations done by individual nodes. The notations in the program are the same as in Section 3.3.3 except that S4 is used instead of $\tilde{S2}$. Since only an individual node is considered, the steps of determining the routing and of additions of links are not essential in this simulation. The program simulates all posssible situations in a node's memory location and all the messages it can receive in each situation. Apriori knowledge allows us not to check several situations.

As a result, this program shows that the Finite-State-Machine acts as is expected, and proves property R7 in Appendix A.

```
MLSM:PROC OPTIONS(MAIN);
N=3;/* NUMBER OF NEIGHBORS */
MA:BEGIN;
/* DECLARATION */
DCL( J,RETURN)BIN FIXED(2C,C);
DCL MG CHAR(4) VARYING;
DCL FI(N) CHAR(5) VARYING;
DCL  FI1(N) CHAR(5) VARYING;
DCL (RI(N),RI1(N)) CHAR (3);
DCL (NIK(N),N1(N),DIK(N),DK1(N),
D_IK(N),M,D,CT,MX1,S,S1,P1,P1,NI, D1, DI,
DI1,DI2) BIN FIXED;
J=C;
/* CONSTANTS */
NODNUM=4; /* NODE COUNTER NUMBER */
INF=50C;  /* THIS IS CONSIDERED AS INFINITE */
D_IK(1)=20; /* MARGINAL DELAY OF LINK (I,1) */
D_IK(2)=10; /* MARGINAL DELAY OF LINK (I,2) */
D_IK(3)=15; /* MARGINAL DELAY OF LINK (I,3) */

/** THE NODE'S MEMORY LOCATIONS (INITIALLY) **/
     FI1(1)='UP'; /* LINK (I,1) IS OPERATIONAL */
     FI1(2)='UP'; /* LINK (I,2) IS OPERATIONAL */
     FI1(3)='UP'; /* LINK (I,3) IS OPERATIONAL */
     DK1(1)=15; /* LAST D RECEIVED ON LINK (I,1) */
     DK1(2)=10; /* LAST D RECEIVED ON LINK (I,2) */
     DK1(3)=5;  /* LAST D RECEIVED ON LINK (I,3) */

     RI1(3)='NIL';

F13:MX1=4;

MX: N1(1)=0;
N11:N1(2)=0;
N12:N1(3)=0;

N13:S1=1;
STA:D1=10;
DEL:RI1(1)='NIL';
SCN1:RI1(2)='NIL';
/* THE MESSAGE */
MG='FAIL';
SCN2:M=2;
NUM:D=10;
DE:    L=1;
NEI:J=J+1;

/* APRIORI KNOWLEDGE ALLOW US NOT TO CHECK THE FOLLOWING */

IF RI1(1)='SCN' & RI1(2)='NIL' THEN GOTO SO2CHA;/*WE HAVE
CHECKED THIS SITUATION */

IF MX1<NODNUM THEN GO TO MXCHA; /*MX1 IS NEVER LESS THAN NI*/
IF MG='MSG' & RI1(L)='SCN' & M<NODNUM THEN GOTO END1;/*IMPO-
SSIBLE BY LEMMA A.1.C */

IF (MX1<N1(1) | MX1<N1(2) | MX1<N1(3)) THEN GOTO N3CHA;

JCHECK=C;
DO K=1 TO N;
IF RI1(K)='SCN' THEN JCHECK=1;
END;
IF S1¬=3 & (JCHECK=0 |D1=INF ) THEN GOTO  SO2CHA;
IF S1=3 & ( JCHECK¬=0 |D1¬=INF ) THEN GOTO  SO2CHA;
IF(JCHECK=0 & D1¬=INF)|(JCHECK¬=0 & D1=INF) THEN GOTO SO2CHA;


IF S1=3 THEN DO; /*THE NODE CAN'T BE IN S3 IF THE FOLLOWING HAPPENS*/
 J4=0;
 DO K=1 TO N;
 IF FI1(K)='UP' & MX1=N1(K) & MX1>NODNUM & DK1(K)¬=INF THEN J4=1;
 END;
IF J4=1 THEN GOTO END1;
END;

IF S1=2 THEN DO;/*THE NODE CAN'T BE IN S2 IF THE FOLLOWING HAPPENS*/
 J1=0; J2=0; J3=0;
DO K=1 TO N;
   IF FI1(K)='UP' & N1(K)¬=NODNUM THEN J1=1;
   IF FI1(K)='UP' & DK1(K)<=D1 THEN J2=1;
   IF RI1(K)='SCN' & DK1(K)=INF THEN J3=1;
 END;
  IF J1=0 & J2=1 & J3=0 THEN GOTO END1;
END;
```

```
IF S1=1 THEN DO;/* THE NODE CAN'T BE IN STATE 1 */
J2=0;
  DO K=1 TO N;
  IF RI1(K)='SCN' THEN IF (N1(K)¬=MX1 | DK1(K)=INF) THEN J2=1;
  END;
 IF J2=0 THEN GOTO END1;
END;
IF S1=2 | S1=4 THEN DO; /* THE NODE CAN'T BE IN THESE STATES */
     J2=0;
  DO K=1 TO N;
  IF RI1(K)='SCN' THEN IF (N1(K)¬=MX1 | DK1(K)=INF) THEN J2=1;
  END;
 IF J2=0 & MX1>NODNUM THEN GOTO END1;
END;


DO K=1 TO N;
 NIK(K)=N1(K); RI(K)=RI1(K); FI(K)=FI1(K); DIK(K)=DK1(K);
END;
MX1=MX1; S=S1; D1=D1 ;N1=NODNUM;

/* THE NODE STARTS TO PROCESS THE MESSAGE */

IF MG='FAIL' THEN DO;
 FI(L)='DOWN';
 CT=0;
 GOTO FSM ;
END;

IF MG='MSG' THEN DO;
 IF FI(L)='READY' THEN FI(L)='UP';
 NIK(L)=M;
 DIK(L)=D+D_IK(L);
 IF D=INF THEN DIK(L)=INF;
 MX1=MAX(M,MX1);
 CT=0;
 GOTO FSM ;
END;


/*****************************************************************/
/******** F I N I T E   S T A T E   M A C H I N E ********/
/*****************************************************************/

        /*********************************************/
        /******** T R A N S I T I O N   1 2 ********/
        /*********************************************/
FSM:
IF S=1 & C1=0 THEN DO;
J1=0;J2=0;
  DO K=1 TO N;
  IF RI(K)='SCN' THEN IF FI(K)¬='UP' THEN J1=1;
  IF RI(K)='SCN' THEN IF (NIK(K)¬=MX1 | DIK(K)=INF)
  THEN J2=1;
  END;
 IF J1=0 & J2=0 THEN DO;
 IF MG='MSG' & M<N1 THEN PUT LIST('CONTRADICTION TO FACT 12')SKIP;
 ELSE DO;
 /*PUT LIST('TRANSITION 12 OCCURED')SKIP;*/
 GOTO UPDATE;
 END;
 GOTO END1;
 END;
END;
   /*****************************************************************/
   /******** T R A N S I T I O N   T O   S T A T E   3 ********/
   /*****************************************************************/
IF (S=1 | S=2 | S=4) & CT=0 THEN DO;
J3=0;
  DO K=1 TO N;
  IF K=L THEN GOTO X3;
  IF RI(K)='SCN' THEN J3=1;
X3:END;
 IF (J3=0 & RI(L)='SCN') & ((MG='MSG' & D=INF)|(MG='FAIL'))
 THEN DO;
  IF MG='MSG' & M<N1 THEN DO;
   IF S=1 THEN PUT LIST('CONTRADICTION TO FACT 13')SKIP;
   IF S=2 THEN PUT LIST('CONTRADICTION TO FACT 23')SKIP;
   IF S=4 THEN PUT LIST('CONTRADICTION TO FACT 43')SKIP;
   GOTO END1;
  END;
/*IF S=1 THEN PUT LIST('TRANSITION 13 OCCURED',J)SKIP; */
/*IF S=2 THEN PUT LIST('TRANSITION 23 OCCURED',J)SKIP; */
/*IF S=4 THEN PUT LIST('TRANSITION 43 OCCURED',J)SKIP; */
```

```
                    /*********************************/
                    /******* C H A N G E   1 ********/
                    /*********************************/
IF S=1 & CT=0 THEN DO;
 JC1=0;
 DO K=1 TO N;
 IF K=L THEN GOTO JCH;
 IF RI(K)='SCN' THEN JC1=1;
JCH:END;
 IF (JC1=1 & RI(L)='SCN') & ((MG='MSG' & D=INF)|(MG='FAIL'))
 THEN DO;
  RI(L)='NIL';
/*PUT LIST('CHANGE 1 OCCURED',J)SKIP;*/ GOTO FSM ;
 END;
END;
       /*********************************************/
       /******** T R A N S I T I O N   2 1 ********/
       /*********************************************/
IF S=2 THEN DO;
 J1=0; J2=0; J3=0;
 DO K=1 TO N;
  IF FI(K)='UP' & NIK(K)¬=NI THEN J1=1;
  IF FI(K)='UP' & DIK(K)<=DI THEN J2=1;
  IF RI(K)='SCN' & DIK(K)=INF THEN J3=1;
 END;
 IF NI=MXI & J1=0 & J2=1 & J3=0 THEN DO;
 IF CT=0 THEN IF MG¬='MSG' THEN GOTO A;
  IF DI=INF THEN DO;
   PUT LIST('CONTRADICTION TO FACT 21')SKIP;
   GOTO END1;
  END;
  ELSE DO;
   DI1=10000;
   DO K=1 TO N;
    IF FI(K)='UP' THEN DO;
     DI2=MIN(DI1,DIK(K));
     NIK(K)=0;
     IF DI2<DI1 THEN PI=K;
     DI1=DI2;
    END;
   END;
   RI(PI)='SCN';
   CT=1;
   S=1;
 /*PUT LIST('TRANSITION 21 OCCURED')SKIP;*/
   GOTO FSM ;
A:END;
 END;
END;
       /*************************************************************/
       /******** T R A N S I T I O N   2 2   O R   4 2 ********/
       /*************************************************************/
IF ((S=2 & CT=0 ) | S=4) & MXI>NI THEN DO;
J1=0;J2=0;
  DO K=1 TO N;
  IF RI(K)='SCN' THEN IF FI(K)¬='UP' THEN J1=1;
  IF RI(K)='SCN' THEN IF (NIK(K)¬=MXI | DIK(K)=INF)
  THEN J2=1;
  END;
 IF J1=0 & J2=0 THEN DO;
/*IF S=2 THEN PUT LIST('TRANSITION 22 OCCURED')SKIP;*/
/*IF S=4 THEN PUT LIST('TRANSITION 42 OCCURED')SKIP;*/
  GOTO UPDATE;
 END;
 END;
          /*********************************************/
          /******** T R A N S I T I O N   2 4 ********/
          /*********************************************/
IF S=2 & CT=0 THEN DO;
 JC1=0;
 DO K=1 TO N;
 IF K=L THEN GOTO ACH;
 IF RI(K)='SCN' THEN JC1=1;
ACH:END;
 IF ((JC1=1 & RI(L)='SCN') & ((MG='ASC' & D=INF)|(MG='FAIL'))) |
 ( RI(L)¬='SUN' & MG='FAIL') THEN DO;
  RI(L)='NIL';
  CT=1;
  S=4;
/*PUT LIST('TRANSITION 24 OCCURED')SKIP; */
  GOTO FSM ;
 END;
END;
```

```
                /**********************************/
                /*******  C H A N G E   4 ********/
                /**********************************/
IF S=4 & CT=0 THEN DO;
 JC1=0;
 DO K=1 TO N;
 IF K=L THEN GOTO JXX;
 IF RI(K)='SCN' THEN JC1=1;
JXX:END;
 IF (JC1=1 & RI(L)='SCN') & ((MG='MSG' & D=INF)|(MG='FAIL'))
 THEN DO;
  RI(L)='NIL';
/*PUT LIST('CHANGE 4 OCCURED',J)SKIP;*/ GOTO FSM;
 END;
END;
          /*****************************************/
          /********  T R A N S I T I O N   3 2 ********/
          /*****************************************/
IF S=3 & MXI>NI THEN DO;
 J4=0;J5=0;
 DO K=1 TO N;
 IF FI(K)='UP' & MXI=NIK(K) & DIK(K)¬=INF
 THEN J4=1;
 IF RI(K)='SCN' THEN J5=1;
 END;
IF J4=1 THEN DO;
  IF J5¬=0 | DI¬=INF THEN DO;
   PUT LIST('CONTRADICTION TO FACT 32')SKIP;
   GOTO END1;
  END;
   DI1=10000;
   DO K=1 TO N;
    IF FI(K)='UP' & NIK(K)=MXI THEN DO;
     DI2=MIN(DI1,DIK(K));
     IF DI2<DI1 THEN FI=K;
     DI1=DI2;
    END;
   END;
  RI(FI)='SCN';
  NI=MXI;
  DI=DIK(FI);
  S=2;
  CT=1;
/*PUT LIST('TRANSITION 32 OCCURED')SKIP;*/
  GOTO FSM ;
  END;
  END;

GOTO END2;

UPDATE:
       DI=10000;
       DO K=1 TO N;
       IF FI(K)='UP' & NIK(K)=MXI THEN DI=MIN(DI,DIK(K));
       END;
       NI=MXI;
       CT=1;
       S=2;
       GOTO FSM ;

FAIL:  DI=INF;
       IF MG='MSG' THEN NI=M;
       RI(L)='NIL';
       S=3;
       CT=1;
       GOTO FSM ;
           /******** C H E C K I N G ********/

 END2:
 IF S=3 THEN DO;
  DO K= 1 TO N;
   IF FI(K)='UP' & NIK(K)=MXI & MXI>NI & DIK(K)¬=INF THEN DO;
   PUT LIST('E R R O R
   THE NODE MUST HAVE LEFT S3',J)SKIP;
   END;
  END;
 END;
```

```
 DO K=1 TO N;
  IF FI(K)='UP' & NIK(K)¬=NI THEN J1=1;
  IF FI(K)='UP' & DIK(K)<=DI THEN J2=1;
  IF RI(K)='SCN' & DIK(K)=INF THEN J3=1;
 END;
 IF NI=MXI & J1=0 & J2=1 & J3=0 THEN DO;
 PUT LIST('E R R O R
 THE NODE MUST HAVE LEFT S2',J)SKIP;
 END;
END;

IF S=4 & MXI>NI THEN DO;
J1=0;J2=0;
  DO K=1 TO N;
  IF RI(K)='SCN' THEN IF FI(K)¬='UP' THEN J1=1;
  IF RI(K)='SCN' THEN IF (NIK(K)¬=MXI | DIK(K)=INF)
  THEN J2=1;
  END;
 IF J1=0 & J2=0 THEN DO;
 PUT LIST('E R R O R
 THE NODE MUST HAVE LEFT S4',J)SKIP;
 END;
END;


END1:
IF L=3 THEN GOTO MESCHA;
  L=L+1;
  GOTO NEI;

MESCHA: IF MG='MSG'  THEN GOTO DCHA;
        MG='MSG' ;
        GOTO DE;

DCHA: IF D=INF  THEN GOTO MCHA;
      D=INF;
      GOTO DE;

MCHA: IF M=6 THEN GOTO SC2CHA;
      M=M+2;
      GOTO NUM;

SC2CHA: IF RI1(2)='SCN' THEN GOTO SO1CHA;
        RI1(2)='SCN';
        GOTO SCN2;

SO1CHA: IF RI1(1)='SCN' THEN GOTO DDCHA;
        RI1(1)='SCN';
        GOTO SCN1;

DDCHA: IF D1=INF THEN GOTO SCHA;
       D1=INF;
       GOTO DEL;

SCHA:  IF S1=4 THEN GOTO N3CHA;
       S1=S1+1;
       GOTO STA;

N3CHA: IF N1(3)>MXI THEN GOTO N2CHA;
       IF N1(3)=6 THEN GOTO N2CHA;
       IF N1(3)=4 THEN N1(3)=6;
       IF N1(3)=0 THEN N1(3)=4;
       GOTO N13;

N2CHA: IF N1(2)>MXI THEN GOTO N1CHA;
       IF N1(2)=6 THEN GOTO N1CHA;
       IF N1(2)=4 THEN N1(2)=6;
       IF N1(2)=0 THEN N1(2)=4;
       GOTO N12;


N1CHA: IF N1(1)>MXI THEN GOTO MXCHA;
       IF N1(1)=6 THEN GOTO MXCHA;
       IF N1(1)=4 THEN N1(1)=6;
       IF N1(1)=0 THEN N1(1)=4;
       GOTO N11;

MXCHA: IF MXI=6 THEN GOTO F3CHA;
       MXI=6;
       GOTO MX;
F3CHA:
ENDMR:END MR;
END MOSF;
```

REFERENCES רשימת מקורות

[BRO 75] C.W. Brown and M. Schwartz, Adaptive Routing in Centralized Computer Communications Netwroks with an Application to Processor Allocation, Proc. IEEE International Conference on Cummunications, San Francisco, June 1975.

[CAN 74] D.G. Cantor and M. Gerla, Optimal Routing in a Packet-Switched Computer Network, IEEE Transactions on Computers, Vol. C-23, No. 10, pp. 1062-1069, October 1974.

[CEG 75] T. Cegrell, A Routing Procedure for the TIDANS Message-Switching Network, IEEE Transactions on Communications, COM-23, No. 6, June 1975, pp. 575-585.

[CHO 72] Routing Strategies for Computer Network Design, Symposium on Computer Communication Networks and Teletraffic, Polytechnic Institute of Brooklyn, April 4-6, 1972.

[DAV 76] Donald W. Davis and Derek L. Barber, Communication Networks for Computers, John Willey & Sons.

[FRAN 71] H. Frank and W. Chou: Routing in Computer Networks, Networks 1 (1971), 99-112.

[FRA 73] L. Fratta, M. Gerla and L. Kleinrock, The Flow Deviation Method : An Approach to Store-and-Forward Communication Network Design, Networks 3, (1973), 97-133.

[GALL 77] R.G. Gallager, A Minimum Delay Routing Algorithm Using Distributed Computation, IEEE Transactions on Communication, COM-25, No. 1 January 1977, pp. 73-85.

[GER 73] M. Gerla, The Design of Store-and-Forward Networks for Computer Communications, Ph.D. dissertation, School of Eng. and Appl. Sci. Univ. of California, LA, Jan. 1973.

[KLEI 64] L. Kleinroch, Communication Nets: Stochastic Message Flow and Delay, McGraw Hill, N.Y. 1964.

[MCQ 77] John. M. McQuillan and David C. Warden, The ARPA Network Design Decisions, Computer Networks 1, 1977, pp. 243-289.

[NAYL 77] W.E. Naylor, A Loop-Free Adaptive Routing Algorithm for Packet Switched Networks, Proc. 4-th Data Communication Symposium, Quebec City, pp. 7.9-7.14, Oct. 1975.

[POU 78] M. Poulos, Simulation of a Minimum Delay Distributed Routing Algorithm, M.S Thesis, M.I.T., Jan. 1978.

[SCAW 77a] M. Schwartz, R.R. Boorstyn and P.L. Pickholtz, Terminal-Oriented Computer Communication Networks, Proceedings of the IEEE, Vol. 60, No. 11, November 1972.

[SCHW 72b] M. Schwartz and C.K. Cheung, The Gradient Projection Algorithm for Multiple Routing in Message-Switched Networks, IEEE Trans. Commun. Vol. COM-24, pp. 449-456, April 1976.

[SEG 77a] A. Segall, Optimal Distributed Routing for Line-Switched Data Networks, EE PUB, No. 306, Technion-Israel Institute of Technology, April 1977.

[SEG 77b] A. Segall, The Modelling of Adaptive Routing in Data Communications Networks, Proc. 1975 National Telecommunication Conference, also in IEEE Transactions on Communications, COM-25, No.1, January 1977, pp. 85-95.

[SEG 77c] A. Segall and P.M. Merlin, A Failsafe Algorithm for Loop-Free Distributed Routing in Data Communication Networks, EE PUB, No. 313, Technion-Israel Institute of Technology, Sept. 1977.

[STE 77] Thomas E. Stern, A Class of Decentralized Routing Algorithms Using Relaxation, IEEE Transactions on Communications , Vol. COM-25, No. 10, October 1977.

[TYM 71] L. Tymes, TYMNET - A Terminal Oriented Communications Networks, AFIPS Conf. Proc. Vol. 38, 1971.